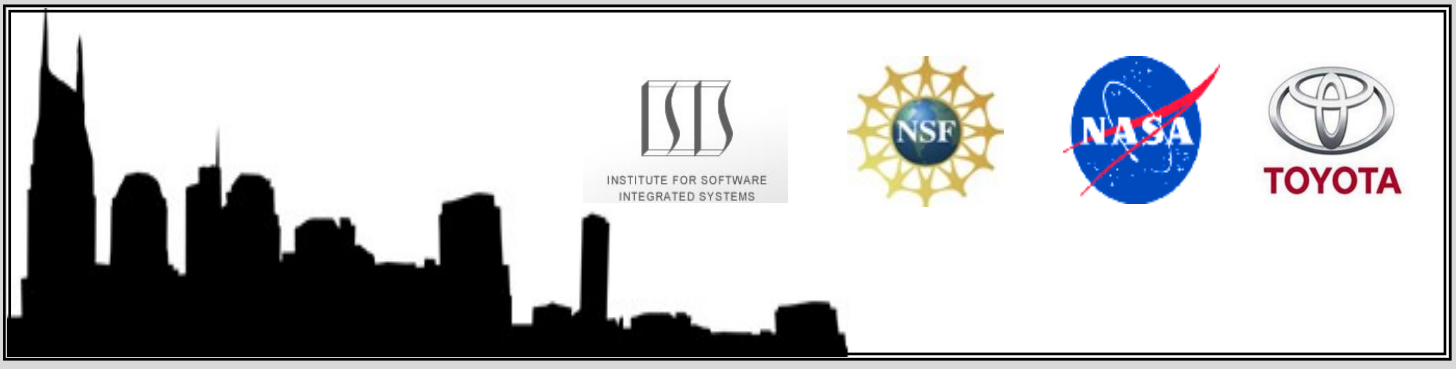
DX-07

18th International Workshop on Principles of Diagnosis

May 29-31, 2007
Nashville, TN, USA

Editors:

Gautam Biswas Xenofon Koutsoukos Sherif Abdelwahed



DX-07

18th International Workshop on Principles of Diagnosis

May 29-31, 2007

Nashville, TN, USA

Gautam Biswas, Xenofon Koutsoukos, and Sherif Abdelwahed, Editors

Foreword

The International Workshop on Principles of Diagnosis is an annual event that started in 1989, rooted in the Artificial Intelligence community. Its focus is on theories, principles and computational techniques for diagnosis, monitoring, testing, reconfiguration and repair of complex systems, and applications of these techniques to real world problems. DX-07 particularly encourages the interaction and the exchange amongst researchers and practitioners from different backgrounds with interests that center around diagnosis and prognosis.

This year we have made significant effort to have a good mix of traditional and non traditional DX researchers and practitioners in the workshop program committee. We have introduced a panel discussion where practitioners will discuss the challenges they face in deploying health management and diagnosis systems in industry today. We changed the format for the reviewing process so the reviewers and the authors would have more opportunities to discuss the content of the papers and the reviews before final decisions were made on the papers. We hope that this produced a more transparent review process, and will lead to more interesting discussions at the Workshop. We really appreciate the additional work that the reviewers and the authors had to put into the extended review process. The result is that we have an excellent set of papers that will be presented at the Workshop: 26 oral presentations and 26 poster presentations.

Many people and organisations have contributed to this workshop.

We would first like to thank all of the authors that have provided the primary material, a set of high quality papers for the workshop. We also thank the Program Committee members for the time and effort they devoted to review the papers and the help they provided in selecting the oral presentation and poster contributions. Thanks also to all of the additional reviewers who helped with the reviewing process. Special thanks to the invited speakers, George Vachtsevanos from Georgia Tech. and Kirby Keller from Boeing Phantom Works, for rounding off an excellent technical program.

This workshop would not have been possible without the support of the National Science Foundation, NASA Ames Research Center, and the Toyota Technical Center. The Embedded & Hybrid Systems program (Helen Gill, Program Director, CISE/CNS) from NSF has provided funding for our invited speakers and student participants from the USA. NASA Ames (Serdar Uckun, Technical Area Lead in Intelligent Systems) and the Toyota Technical Center (Liu Qiao, GM and Chief Technologist) have provided support for various miscellaneous expenses.

Many thanks to the local organisation team at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University for their excellent administrative support in organizing the Workshop: Kristy Fisher, Kyl Boggs, and Michele Codd. Special thanks to Matthew Daigle and Indranil Roychoudhury for their excellent work in managing the DX-07 web page, the online submission system, and for being in the frontline for handling all of your questions during the submission and reviewing processes.

We do hope that all of you will find DX-07 interesting with stimulating presentations and discussions as well as enjoyable social events. In addition, please do not forget to enjoy the best that Music City has to offer.

Gautam Biswas, Xenofon Koutsoukos, and Sherif Abdelwahed
ISIS, Vanderbilt University
Nashville, TN 37235, USA
May 2007

Workshop Organization

Program Co-Chairs

Gautam Biswas	Vanderbilt University, USA
Xenofon Koutsoukos	Vanderbilt University, USA
Sherif Abdelwahed	Vanderbilt University, USA

International Program Committee

Carlos J Alonso	Univ. de Valladolid, Spain
Tom Brotherton	Intelligent Automation Corporation, USA
Luca Console	University of Torino, Italy
Marie-Odile Cordier	University of Rennes1/IRISA, France
Philippe Dague	University of Paris South, France
Johan de Kleer	PARC, USA
Teresa Escobet	Politechnic University of Catalonia, Spain
Eric Fabre	IRISA, France
Christoforos Hadjicostis	University of Illinois at Urbana Champaign, USA
Michael Hofbaur	Graz University of Technology, Austria
Nagarajan Kandasamy	Drexel University, USA
Gabor Karsai	Vanderbilt University, USA
Stephane Lafortune	University of Michigan, USA
Jan Lunze	Ruhr-Universität Bochum, Germany
Pieter Mosterman	The MathWorks, Inc., USA
Sriram Narasimhan	Univ. of California, Santa Cruz & NASA Ames, USA
Mattias Nyberg	Scania CV Sodertalje, Sweden
Ann Patterson-Hine	NASA Ames, USA
Krishna Pattipati	University of Connecticut, USA
Gregory Provan	Cork College, Ireland
Belarmino Pulido	Univ. de Velladolid, Spain
Liu Qiao	Toyota Technical Center, USA
John Sheppard	John Hopkins University, USA
Marcel Staroswiecki	Lille University, France
Peter Struss	Technical University of Munich, Germany
Markus Stumptner	University of South Australia, Australia
Louise Travé-Massuyès	LAAS-CNRS, France
Timothy Wilmering	Boeing PhantomWorks, USA
Franz Wotowa	Graz University of Technology, Austria

Additional Reviewers

Eleftheria Athanasopoulou	University of Illinois at Urbana-Champaign, USA
Joao Carlos Basilio	University of Michigan, USA
Matthew Daigle	Vanderbilt University, USA
Elodie Chanthery	LAAS-CNRS, France
Oskar Dressler	OCC'M Software, Germany
Daniele Theseider Dupré	Universita' del Piemonte Orientale, Italy
Carine Jauberthie-Salsmann	LAAS-CNRS, France
John Kinnebrew	Vanderbilt University, USA
Edouard Diez Lledo	LAAS-CNRS, France
Ole J. Mengshoel	NASA Ames Research Center, USA
Abbas Moustafa	Vanderbilt University, USA
Claudia Picardi	Universita' di Torino, Italy
Juan J Rodrigues Diez	Universidad de Burgos, Spain
Indranil Roychoudhury	Vanderbilt University, USA
Yu Ru	University of Illinois at Urbana-Champaign, USA
Anooshiravan Saboori	University of Illinois at Urbana-Champaign, USA
Audine Subias	LAAS-CNRS, France
Shreyas Sundaram	University of Illinois at Urbana-Champaign, USA
George Takos	University of Illinois at Urbana-Champaign, USA
Thierry Vidal	IRISA/INRIA, France
Yin Wang	University of Michigan, USA

Workshop Organizing Committee

Sherif Abdelwahed	Vanderbilt University, USA
Gautam Biswas	Vanderbilt University, USA
Kyl Boggs	Vanderbilt University, USA
Matthew Daigle	Vanderbilt University, USA
Kristy Fisher	Vanderbilt University, USA
Xenofon Koutsoukos	Vanderbilt University, USA
Indranil Roychoudhury	Vanderbilt University, USA

Table of Contents

Invited Talks

Industry Panel

Papers

Posters

Invited Talks

Health Management Technology Integration

Kirby Keller

Boeing Phantom Works

Abstract

Academia, industry and government research laboratories have made great strides in developing and maturing software and hardware technology to monitor and even predict the health of machines. The output of health monitoring (i.e., diagnosis, prognosis, usage) are further processed and integrated with machine controls, operations, maintenance and logistics to produce a health management capability with the end goal of enabling business to better utilize their assets. This presentation examines some of the challenges or hurdles to the integration of health monitoring/management technology into aircraft fleet operations. Key focus areas or points of integration include creating the business case, consideration of health management in the design process, integrating health monitoring hardware and software into the aircraft, leveraging the resulting output into the operations and support decision loop, handling the additional data, addressing regulatory concerns, and the allocation of the responsibilities and roles of the parties involved in the development, use and maturation of a health management system. Boeing Phantom Works Support and Services thrust is tasked with facilitating the maturing health management technology for integration into Boeing products. The presentation also describes some of the efforts by the Phantom Works to address the health management integration challenges above including the creation of a Health Management Engineering Environment. This environment consists of three laboratories: Program Analysis and Modeling to establish the business case, Development Laboratory to provide the tools and processes, and the Operations Laboratory to perform integrated demonstrations in realistic environments.

Biography

Dr. Kirby Keller is a Technical Fellow for the Boeing Company. He has 33 years experience in the development of mission and vehicle health management systems to air, ground and space vehicles. His current duties include Technical Lead for the Integrated Vehicle Health Management programs within the Support and Services Thrust of Boeing Phantom Works. In this position, he is the principal investigator and system designer for several internal research and development projects that are focused on developing a Boeing wide common architecture and best practices for IVHM. He is program manager for the USAF sponsored Dual Use Science and Technology (DUST) Aircraft Electrical Power System Prognostics and Health Management (AEPHM) program. He was the technical lead for the Navy Open System Architecture for Condition Based Monitoring research program and for the USAF/Boeing Unmanned Combat Air Vehicle (UCAV) System prognostics and health management risk reduction demonstrations. Other, past programs activity includes the On-line Flight Control Diagnostic System sponsored by the Navy and U. S. Army Rotorcraft Pilot's Associate program. He holds a PhD in Mathematics from Iowa State University and is the author of over 40 technical papers, conference presentations, and technical reports.

An Integrated Architecture for Fault Diagnosis and Failure Prognosis with an Application to Aircraft Systems

George Vachtsevanos

Georgia Institute of Technology

Abstract

This presentation is introducing an integrated fault diagnosis and failure prognosis methodology as applied on-line to flight critical aircraft systems. Enabling technologies include vibration and flight regime data pre-processing, de-noising algorithms to improve signal to noise ratio, feature or condition indicator selection and extraction routines, and model-based fault diagnosis and failure prognosis algorithms for accurate detection and robust prediction of the remaining useful life of failing components. We elaborate on the design of degradation or fatigue models using finite element analysis and nonlinear dynamic models in the frequency domain in order to gain a better understanding of the physics of failure mechanisms, to extract an optimum feature vector and to establish a basis for incipient failure detection and prediction. The modeling framework is coupled with Bayesian estimation techniques, specifically particle filtering, for accurate diagnosis and prognosis. Seeded fault data from a testcell and fleet aircraft data are used to train and validate the algorithms. A Graphical User Interface and computer code for all processing algorithms are implemented on conventional health monitoring apparatus available on several military aircraft. Results from "blind" testing demonstrate the efficacy of the architecture.

Biography

Professor Vachtsevanos was born in Kozani, Greece. He attended the City College of New York and received his B.E.E. degree in 1962. He received an M.E.E. degree from New York University and his Ph.D. degree in Electrical Engineering from the City University of New York in 1970. His research focused on adaptive control systems. After graduate school, he taught within the City University System of New York from 1970 through 1975; from 1975 through 1977 he served as an Associate Professor of Electrical Engineering at Manhattan College. In 1977 he was elected to a chair professorship in Electrical Engineering at the Democritus University of Thrace, Greece, where he served as the Pro-Rector during the 1978-79 academic year. He joined the Georgia Institute of Technology as a Visiting Professor in 1982-1983 and as a Professor in 1984. He is also serving as an adjunct faculty member in the School of Textile and Fiber Engineering. He was the 3M McKnight Distinguished Visiting Professor at the University of Minnesota, Duluth during the 1994 Spring Semester. Since joining the faculty at Georgia Tech, Dr. Vachtsevanos has been teaching courses and conducting research on intelligent systems, robotics and automation of industrial processes and diagnostics/prognostics of large-scale complex systems.

Industry Panel

Primary challenges in deploying health monitoring and fault diagnosis systems in industry today:

How do we bridge technology gaps and impediments to successful deployment?

Panelists: Christophe Dousson, Stan Ofsthun, Liu Qiao, and Serdar Uckun

Moderator: Johan de Kleer

Description

This panel aims at bringing together leading researchers and developers from some of the industries where fault diagnosis, health monitoring, and fault-tolerance are critical for successful deployment and safety of systems. The panelists will debate the following issues:

- Real challenges and problems in health monitoring and fault diagnosis that their industries face today.
- Promising technologies and gaps in the current state of the art that prevent successful applications.
- Business and economic obstacles.
- Suggestions on how these obstacles can be overcome. What realistic contributions can the research community make? How can the academic research community work with industrial practitioners to develop useful and successful solutions?

Idea Takeaway

Each panelist will describe one or two compelling challenge problems in health monitoring and fault diagnosis that their industry faces. The audience will learn what technical and business obstacles stand in the way of addressing these problems, and what the real gaps in technology are. The ensuing discussion will look for realistic and specific ways in which some of the hard problems can be jointly addressed, and whether a good collaborative model can be established, which respects the intellectual property issues that face industry as well as academia's need to develop publish research results.

Statements and Biographical Information of Panelists

Christophe Dousson - France Telecom

Diagnostic Challenges

In the last decades, the fields of a telecom operator like France Telecom have changed: the main objective is not only to deploy and maintain a telecom network but to propose and to guarantee more and more innovative telecom services. The "Diagnosis" team mainly contributes to satisfy end-customers in two areas: the service supervision and the customer hotline. (1) *Services supervision*: the objective is to collect and synthesize a huge amount of information coming from telecoms equipment (routers, probes...) in order to make them manageable by human operators. Purposes of this supervision could be QoS reporting, intrusion detection, fault detection...The main difficulties raise from the high number of sensors and the inherent distributed architecture but also from the knowledge acquisition since the system evolves continuously. Our approach based on chronicle recognition will also be evoked. (2) *Customer hotline*: the objective is to give a diagnosis tool which can be used by customer and/or by technical support in order to satisfy as soon as possible the customer. One hard point is to give an "intuitive" modeling in order to be able to update the system with new services and/or problems but it is also required to explain the diagnosis in order to convince the end user. As this challenge is more recent for us, just some clues on what could be done to achieve that will be given to start panel discussion.

Biography

After preparing his PhD at LAAS/CNRS (Toulouse) and graduated of the Ecole Polytechnique, Dr. Christophe Dousson joined France Telecom R&D in 1994 on a research position in Artificial Intelligence applied to Diagnosis. He was firstly involved in the field of fault management of telecommunications networks. Nowadays, the applicative fields are more related to virus and intrusion detection but the main

topic of his work remains temporal reasoning for supervision and situations recognition. He also addressed the problem of automatic or assisted chronicle discovering based on log analysis and also based on a behavioral model of the system. Since 2004, he is also the head of the "Diagnosis" research team.

Stan Ofsthun - Boeing Phantom Works

Diagnostic Challenges

In Boeing's product lines span commercial and military aerospace vehicles, weapon systems, and systems of systems that are used around the world, by many diverse organizations, in a variety of harsh environments. Each of these Boeing products requires an optimized combination of autonomous embedded diagnostic capabilities to support real time operational decisions and near real time off board diagnostic capabilities for effectively driving maintenance decisions. The specific technical diagnostic problems that are encountered on individual programs are too numerous to discuss in this panel format. Rather, Boeing will describe the systemic process oriented diagnostic problems that are common across many platforms, and that can potentially be addressed by technologies developed in industry and academia. Representative problem areas include inefficient and inconsistent analytical processes, gaps between these analytical processes and the diagnostic design processes, and the lack of an effective process for maturing the diagnostic implementation over the product life cycle.

Biography

Stan Ofsthun is a Technical Fellow in the Boeing Phantom Works organization. His career has primarily focused on the practical application of Testability, Built In Test, Integrated Diagnostics, and Integrated Vehicle Health Management (IVHM) technologies, processes, and tools to a variety of Boeing platforms, including F-15, F/A-18, X-45C and various space/weapons programs. He has also developed methodologies to integrate these efforts with reliability and safety disciplines, and is currently serving as the St Louis site technical representative on Boeing's Reliability, Maintainability, and Systems Health (RM&SH) steering team. His educational background includes a B.S. in Electrical Engineering (Rensselaer Polytechnic Institute), a M.S. in Electrical Engineering (Washington University), and graduate certificates in Artificial Intelligence and Web Content Design (Washington University).

Liu Qiao - Toyota Technical Center

Diagnostic Challenges

Automotive electronics and software represent a continuously increasing share in the added value of automotive products. Up to 90% of all automotive innovations are related to electronics systems. Increasing demands in vehicle safety, driver assistance, and comfort drives this trend. In addition legal requirements or environmental needs can only be fulfilled by using electronic hard- and software extensively. All vehicle domains are affected. As a consequence, the complexity of automotive electronics/system diagnosis is growing exponentially. The challenges also come from shortened development cycles, market competition and increasing customer expectation. We may have to reconsider the way of diagnosis.

Biography

Dr. Liu Qiao is General Manager and Chief Technologist of Technical Research Dept. He is responsible for electrical, electronics and vehicle control related research at Toyota Technical Center, Toyota Motor Engineering and Manufacturing North America. Switching from a university faculty position, Dr. Qiao started his automotive career as an advanced automotive control system expert, advanced technology manager, eBusiness manager and research manager. Dr. Qiao successfully led Toyota's Canadian hybrid vehicle project and its market introduction. Dr. Qiao received Electrical Engineering Ph.D. from Tohoku University in Japan. He is an active member/supporter of many academic associations.

Serdar Uckun - NASA Ames Research Center

Diagnostic Challenges

NASA designs, develops, and operates unique, highly customized spacecraft in uncharted territories. These spacecraft often experience failures due to operations outside design margins, extended exposure to extreme environments, and unforeseen interactions between the spacecraft and the external environment. Space operations provide a rich application domain for diagnostic technologies. However, there are several challenges that limit the applicability of cutting-edge diagnostic tools to space operations. These include: 1) verification and validation hurdles; 2) mismatch between computational requirements of advanced diagnostic

algorithms versus the capabilities of space-qualified computing platforms; 3) lack of autonomous operations concepts for optimal decision making under uncertainty; and 4) requirements for extremely low false-positive and false-negative detection rates (especially in human space flight applications).

Biography

Dr. Serdar Uckun is the Technical Area Lead for Discovery and Systems Health in the Intelligent Systems Division at NASA Ames Research Center. The technical area consists of approximately 70 researchers and engineers and focuses on engineering and scientific data understanding problems pertinent to NASA, including Integrated Systems Health management (ISHM). The technical area has the single largest ISHM R&D and systems engineering team at NASA, with over 50 people involved in ISHM-related tasks funded by NASA's Exploration Systems, Aeronautics, and Space Operations Mission Directorates. In addition to his line management role, Dr. Uckun has been serving as the Project Manager for the ISHM project under NASA's Exploration Technology Development Program. He participates in various NASA planning and strategy activities as a subject matter expert on health management. He has an M.D. from Ege University in Izmir, Turkey, M.S. in Biomedical Engineering from Bogazici University in Istanbul, Turkey, and a Ph.D. in Biomedical Engineering from Vanderbilt University in Nashville, TN. His technical interests include monitoring, diagnosis, prognostics, and scheduling. He has over thirty publications in peer-reviewed journals and conferences.

Johan de Kleer – Palo Alto Research Center

Biography

Johan de Kleer is a Principal Scientist in the Embedded Reasoning Area in PARC's Intelligent Systems Laboratory. His core interest is building a system which can reason about the physical world as well as he can. Until recently, he was Laboratory Manager of PARC's Systems and Practices Laboratory of Xerox's Palo Alto Research Center. This interdisciplinary laboratory conducted research ranging from social science to robotics. Two foci of the laboratory were: (1) Smart Matter - which exploits trends in miniaturization and integration to create a new generation of products and processes that benefit from the coupling of computational and physical worlds, and (2) Knowledge - knowledge management, which includes social science research on organizations, knowledge representation and understanding images and video streams. Johan received his Ph.D. from M.I.T. in 1979 in Artificial Intelligence. He has published widely on Qualitative Physics, Model-Based Reasoning, Truth Maintenance Systems, and Knowledge Representation. He has co-authored three books: Readings in Qualitative Physics, Readings in Model-Based Diagnosis, Building Problem Solvers. In 1987 he received the prestigious Computers and Thought Award at the International Joint Conference on Artificial Intelligence. He is a fellow of the American Association of Artificial Intelligence and the Association of Computing Machinery.

Papers

Online Posterior Probability Calculation for Failure Diagnosis in Finite State Machines based on Unreliable Sensor Information

Eleftheria Athanasopoulou and **Lingxi Li** and **Christoforos N. Hadjicostis**
Department of Electrical and Computer Engineering
and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street, Urbana, IL 61801–2307
{athanaso, lingxili, chadjic}@uiuc.edu

Abstract

In this paper we develop an efficient recursive algorithm to calculate the probability that an observed, possibly corrupted sequence was produced by either a fault-free or a faulty candidate finite state model. Our ultimate goal is to evaluate how well a given model matches the observations, keeping in mind that the observations themselves may be corrupted due to sensor malfunctions or other unreliabilities. By performing a recursion in the number of observation steps, the optimal solution is reached in a memory and computationally efficient fashion. The recursive algorithm can be used for online monitoring and provides, at each observation step, the posterior probability of a model given the possibly corrupted observations (obtained by the unreliable sensors). The proposed methodology is illustrated via an application to the problem of failure diagnosis for a communication protocol.

Index terms: Finite state machines, discrete event systems, hidden Markov models, sensor failures, failure diagnosis, maximum posterior probability, model classification.

Introduction

Failure diagnosis is an important aspect of modern systems and networks, particularly in applications that are life-critical and require high reliability. In this work we focus on diagnosis based on discrete event system (DES) formulations, i.e., we consider failure diagnosis in systems whose state space is discrete and their evolution is event-driven. Any large-scale dynamic system, such as a computer system, a manufacturing system or a chemical process can be modeled as a DES at some level of abstraction. Much work has been done in failure diagnosis of discrete event systems (DESs) including deterministic diagnosis (Lin 1994), (Sampath *et al.* 1995), (Zad, Kwong, & Wonham 2003), (Wu & Hadjicostis 2005), probabilistic diagnosis (Hadjicostis 2005), (Athanasopoulou, Li, & Hadjicostis 2006), and diagnosis in stochastic finite automata (Blanke *et al.* 2003), (Thorsley & Teneketzis 2005).

In this paper, we consider a formulation similar to the one introduced in our previous work on probabilistic failure diagnosis under unreliable observations (Athanasopoulou, Li, & Hadjicostis 2006). More specifically, we are given two known deterministic finite state machine (FSM) models (one corresponding to the fault-free version of the underlying system and the other corresponding to a given faulty version of the system), and we aim to determine which of the two competing models has more likely produced a given observed sequence. We assume that the input sequence that drives each model is unknown but the prior input probability distribution is known. Sensors are used to observe the system under diagnosis via its outputs, which are associated with transitions in the FSM model. However, the information that the sensors provide may be corrupted due to various reasons, such as inaccurate measurements, limited resolution, degraded sensor performance due to aging, or hardware failures. As a result, the observed sequence may not resemble the actual output sequence. We consider unreliable sensors that may cause outputs to be deleted, inserted, substituted or transposed with certain known probabilities.

In order to choose among a set of competing models the one that best matches the observations, we need to evaluate how well each model matches the observed sequence. Given a model and an observed sequence, the (standard) evaluation problem consists of computing the probability that the observed sequence was produced by the model. If the observed sequence was not corrupted, its probability given a particular model could be calculated as the sum of the probabilities of all possible state sequences that are consistent with the observations. This can be done online by taking advantage of the graphical structure of the model (Jordan 2004) to use a recursive algorithm similar to the forward algorithm (Rabiner 1989), which solves the evaluation problem in hidden Markov models (HMMs) (e.g., in (Rabiner 1989), (Poritz 1988), (Ephraim & Merhav 2002)) and is used in various pattern recognition applications (Vidal *et al.* 2005).

Under unreliable sensors, several output sequences may correspond to the observed sequence; in fact, we need to identify all possible output sequences and then compute the probability with which the output sequences agree with *both* the underlying FSM model and the observations (also allowing, of course, for sensor unreliabilities). For example, if the observed sequence is corrupted with output deletions, the standard forward algorithm is insufficient because there are potentially infinite output sequences that agree with the observed sequence. In this paper, we develop an algorithm that computes the probability of the possibly corrupted observed sequence given the system model in a recursive, computationally efficient manner. The proposed algorithm relates to (and generalizes) recursive algorithms for the evaluation

problem in HMMs and the parsing problem in probabilistic automata.

A first attempt to address the inability of the standard forward algorithm to handle sensor failures (that lead to vertical loops in the associated trellis diagram) appeared in our earlier paper (Athanasopoulou, Li, & Hadjicostis 2006). In that earlier work we first constructed an observation FSM (that captures all possible output sequences) and then operated on a composition of the observation FSM with each candidate model. However, that algorithm has high computational complexity and cannot be used in online monitoring schemes. In this paper, we propose a recursive algorithm that allows us to efficiently compute the probability that a certain FSM matches the observed sequence online: each time a new observation is obtained, the algorithm simply updates the information it keeps track of in order to calculate the probability that a given model has produced the (possibly corrupted) sequence that has been observed so far.

Preliminaries

FSM and Markov Chain Notation

A deterministic FSM model S can be described by a six-tuple $S = (Q, X, Y, \delta, \lambda, q_0)$, where Q is the finite set of states (without loss of generality we will also denote each state by its index, i.e., $Q = \{1, 2, ..., |Q|\}$, where $|Q|$ denotes the size of Q); X is the finite set of inputs; Y is the finite set of outputs; δ is the state transition function; λ is the output function; and q_0 is the initial state (Cassandras & Lafortune 1999). The state $q[m+1]$ of the FSM at time epoch $m+1$ is specified by its state $q[m]$ at time epoch m and its input $x[m+1]$ via the state transition function δ as $q[m+1] = \delta(q[m], x[m+1])$. The output of the FSM is associated with the FSM transition and is specified via the output function λ as $y[m+1] = \lambda(q[m], x[m+1])$. The FSMs we consider here are event-driven and we use m to denote the time epoch between the occurrence of the m^{th} and $(m+1)^{th}$ input.

For simplicity, we assume that the inputs applied to a given FSM are statistically independent from one time step to another and that their probability distribution is fixed so that, at any given time step m, from a given state each input takes place with fixed probability (other types of input distributions can be handled in a similar fashion by enlarging the state space). Since the FSM makes a transition to the next state depending on the value of the present state *and* input, it behaves as a homogeneous Markov chain (Kemeny, Snell, & Knapp 1976); the corresponding Markov chain can be obtained from the FSM by assigning to each transition a probability that depends on the probabilities of the inputs that cause it. Since we have no access to the inputs that drive the FSM but we only observe the outputs it produces, the FSM state sequence is not completely known and the resulting system is a hidden Markov model (HMM). An HMM is described as a five-tuple $(Q, Y, \Delta, \Lambda, \pi[0])$, where Q is the set of states, Y is the set of outputs, Δ captures the state transition probabilities, Λ captures the output probabilities associated with transitions, and $\pi[0]$ is the initial state probability distribution vector.

Example: The FSM S with state transition diagram as shown on the left of Figure 1 will be used as a running example throughout the paper. S has four states $Q = \{1, 2, 3, 4\}$, three inputs $X = \{x_1, x_2, x_3\}$, and three outputs $Y = \{a, b, c\}$. Each transition is labeled as $x_i \mid \sigma$, where $x_i \in X$ denotes the input that drives the FSM and $\sigma \in Y$ denotes the associated output produced by the FSM. If we assign equal prior probabilities to the inputs, i.e., if each input has probability $1/3$ of occurring, the resulting HMM is shown on the right of Figure 1. Each transition in the HMM is labeled as $p \mid \sigma$, where p denotes the probability of the transition and $\sigma \in Y$ denotes the output produced. □

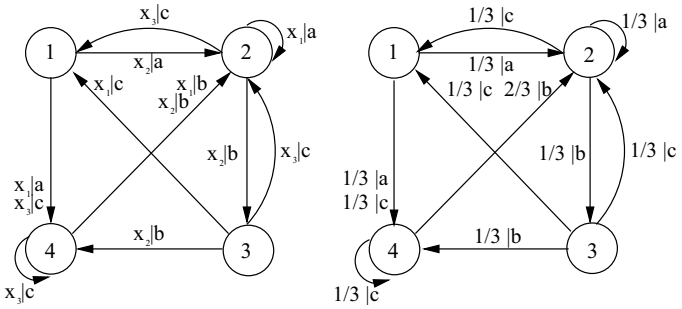


Figure 1: State transition diagram of FSM S in our example (left) and its corresponding HMM (right).

Posterior Probability Calculation with Uncorrupted Observations (Reliable Sensors)

As mentioned in the introduction, our objective is to compare the probability that the FSM under consideration is fault-free against the probability that the FSM is faulty, given the observed sequence and assuming known input probability distributions and initial state probability distribution. In this section, we examine the simplest case where no sensor failures are present. It will become apparent later that this case does not involve any complications such as loops in the trellis diagram and the calculations can be easily performed recursively by a forward-like algorithm.

To minimize the probability of incorrect diagnosis, we use the maximum *a posteriori* probability (MAP) rule, i.e., we compare $P(S_1 \mid z_1^L) \gtrless P(S_2 \mid z_1^L)$, where S_1 represents the fault-free FSM, S_2 represents the faulty FSM, and z_1^L represents the observed sequence $< z[1], z[2], ..., z[L] >$. In this case of reliable sensors, the observed sequence z_1^L is equal to the actual output sequence y_1^L, i.e., $z_1^L = y_1^L =< y[1], y[2], ..., y[L] >$. Clearly, if the probability of the fault-free machine given the observed sequence is larger than the probability of the faulty machine given the observed sequence we should declare that the machine is fault-free, otherwise we should declare that the machine is faulty.

Assuming known prior probabilities for the fault-free and faulty FSMs given by P_1 and P_2 respectively (with $P_1 + P_2 = 1$), the above comparison can be reduced to $P(z_1^L \mid S_1) \cdot P_1 \gtrless P(z_1^L \mid S_2) \cdot P_2$. To calculate the probability $P(z_1^L \mid S)$ of the observed sequence given a particular model S (where S could be either S_1 (fault-free) or S_2 (faulty)), first we capture the evolution of S as a function of time for a given observed

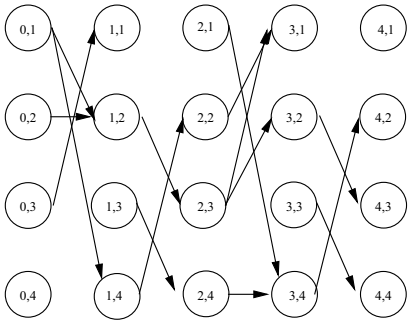


Figure 2: Trellis diagram corresponding to FSM S in our example (transition probabilities are not included for clarity).

sequence by constructing the trellis diagram of FSM S. The state sequences that agree with the observed sequence (consistent sequences) are those that start from any valid initial state and end to any final state while ensuring that the output at each step m matches the observed output $y[m]$. Due to the Markovian property, the probability of a specific consistent state sequence can be easily calculated as the product of the initial state probability and the state transition probabilities at each time step. Thus, to calculate the probability $P(z_1^L \mid S)$ we need to first identify all consistent sequences and their probabilities and then sum up the total probability. The computation of $P(z_1^L \mid S)$ is not recursive on the number of observation steps, hence it is not amenable for online monitoring.

In order to make the computation recursive we can use a forward-like algorithm. More specifically, let $\mathscr{A}_\sigma$ be a $|Q| \times |Q|$ matrix whose (j^{th},k^{th}) entry captures the transition probability from state k to state j that produces output $\sigma \in Y$. Let the initial probability distribution be known and denoted by $\rho[0]$, i.e., a $|Q|$-dimensional vector, whose j^{th} entry denotes the probability that S is initially in state j), the joint probability of the state at time step m and the observed sequence y_1^m is captured by the vector $\rho[m]$ where the entry $\rho[m](j)$ denotes the probability that the machine is in state j at step m and $y[1],\ldots,y[m]$ have been observed. When $y[m+1]$ becomes available at step $m+1$, we can update the joint probability of the state of the HMM and the observed sequence y_1^{m+1} as

$$\rho[m+1] = \mathscr{A}_{y[m+1]}\rho[m], \;\; m = 0,1,...,L-1.$$

If the state probabilities at the last observation step L are captured by $\rho[L]$, the probability that the observations were produced by FSM S is equal to the sum of the elements of $\rho[L]$, i.e., $P(z_1^L \mid S) = \sum_{j=1}^{|Q|} \rho[L](j)$. Using this recursive algorithm we only need to store $|Q|$ values at each step, namely the probabilities of the current state being $j \in \{1,2,...,|Q|\}$ for the particular sequence we have observed up to that step.

Example (continued): Suppose that we monitor the FSM under diagnosis for $L = 4$ steps and we observe the sequence $z_1^4 =< abcb >$. The trellis diagram for FSM S of Figure 1 is shown in Figure 2 (transition probabilities are not shown for clarity of presentation and pairs of states that are not connected are associated with zero transition probabilities). Each state of the trellis diagram is identified by a pair (m, j), where $m = 0,1,2,...,L$ denotes the observation step and $j \in \{1,2,...,|Q|\}$ denotes the state of S. For example, the probability of a transition from state $(0,1)$ to state $(1,4)$ producing output a is $1/3$. Assuming uniform initial distribution, the probability that the observations were produced by S can be calculated recursively and is given by $P(z_1^4 \mid S) = 0.0062$. Note that this probability is expected to go down as the number of observations increases; there are various ways to renormalize the values but we will not discuss them in this paper. □

Posterior Probability Calculation with Corrupted Observations (Unreliable Sensors)

Now we consider the more interesting scenario where sensor failures may convert the output sequence to a corrupted observed sequence.

Sensor Failure Model

The output sequence $y_1^{L_y} =< y[1],y[2],...,y[L_y] >$ produced by the system under diagnosis may become corrupted due to sensor unreliabilities; this implies that the length of the observed sequence z_1^L will generally be different from the length of the output sequence (i.e., $L \neq L_y$). We consider sensor failures that are transient and occur independently at each observation step with certain (known) probabilities that could *depend* on the observation step, e.g., the probability of some transient errors could vary as a function of time. We also make the reasonable assumption that, conditioned on the observed sequence, sensor failures are independent of the inputs that drive the system under diagnosis.

The sensor failures we consider may result in the deletion, insertion, substitution or transposition of adjacent outputs. If an output $\sigma \in Y$ is deleted by the sensor, then we do not observe an output, i.e., the deletion causes $\sigma \rightarrow \varepsilon$, where ε denotes the empty label. Similarly, if an output $\sigma \in Y$ is inserted by the sensor, then we observe σ instead of ε, i.e., the insertion causes $\varepsilon \rightarrow \sigma$. Also, if an output $\sigma_j \in Y$ is substituted by $\sigma_k \in Y$, then we observe σ_k, i.e., the substitution causes $\sigma_j \rightarrow \sigma_k$. Finally, the corruption of sequence $< \sigma_j\sigma_k >$ to $< \sigma_k\sigma_j >$ is referred to as a transposition error.

To perform the posterior probability calculation, we construct the trellis diagram of FSM S as before but modify it to capture the sensor failures. From now on, we call each column of the trellis diagram a *stage* to reflect the notion of an observation step. Deletions appear in the trellis diagram as vertical transitions within the same stage and insertions appear as one-step forward transitions. A substitution appearing at a particular observation step results in a change of the transition functionality of the FSM for that step. The transposition of two adjacent outputs appears in the trellis diagram as an erroneous transition that spans two columns.

Given the sensor failure model, we can assign probabilities to all types of errors on the observed sequence. Since the probabilities of sensor failures and inputs are known and are (conditionally) independent, we can easily determine the probabilities associated with transitions in the trel-

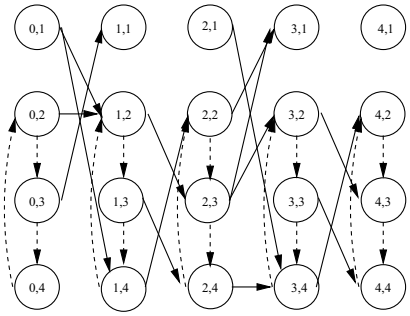


Figure 3: Trellis diagram corresponding to FSM S in our example with deletions (transition probabilities are not included for clarity).

lis diagram. Due to space limitations, we focus on deletions which is the most challenging case of sensor failures because they may produce vertical cycles in the trellis diagram. We assume that a deletion $d_{\sigma'}$ of output σ' occurs with known probability $p_{d_{\sigma'}}[m+1]$ at observation step $m+1$ when S is in a state from which a transition that outputs σ' is possible. Let $D = \{d_{\sigma_1}, d_{\sigma_2}, ..., d_{\sigma_{|D|}} \mid \sigma_1, \sigma_2, ..., \sigma_{|D|} \in Y\}$ be the set of deletions and define a function *out* to allow us to recover the corresponding output in the set Y given a deletion, i.e., $out(d_{\sigma'}) = \sigma'$. When constructing the trellis diagram, we assign probabilities to the transitions as follows: (i) each forward (normal) transition from state (m, j) to state $(m+1, k)$ associated with output σ is assigned probability $(1 - \sum_{\substack{\forall d_{\sigma'} \in D \text{ s. t.} \\ \delta(j, out(d_{\sigma'})) \neq \emptyset}} p_{d_{\sigma'}}[m]) \cdot \mathscr{A}_{\sigma}(k, j)$, where $(1 - \sum_{\substack{\forall d_{\sigma'} \in D \text{ s. t.} \\ \delta(j, out(d_{\sigma'})) \neq \emptyset}} p_{d_{\sigma'}}[m])$ is the probability that no deletion (possible from state j) occurs and where $\mathscr{A}_{\sigma}(k, j)$ is the probability of going from state j to state k while producing output σ, (ii) each vertical transition (corresponding to deletions) from state (m, j) to state (m, k) is assigned probability $\sum_{\substack{\forall d_{\sigma'} \in D \text{ s. t.} \\ \delta(j, out(d_{\sigma'})) = k}} (p_{d_{\sigma'}}[m] \cdot \mathscr{A}_{\sigma'}(k, j))$. Note that other ways of obtaining these probabilities are possible, e.g., under different models of sensor failures. What is important (and a challenge) here are not the values of probabilities but the structure of the trellis diagram (in particular the loops that are present).

Example (continued): In FSM S of our example suppose that deletion of b may occur and that the observed sequence is $z_1^4 =< abcb >$. The output sequence $y_1^{L_y}$ could be of any length; examples of possible output sequences are $< abbbcb >$, $< babcb >$, and $< babbcbbb >$. The resulting trellis diagram is shown in Figure 3, where dashed arcs represent erroneous transitions. In this example, we assume that the deletion probability p_{d_b} is fixed for all observation steps. The probabilities of transitions are not shown in the figure for clarity, but they can be computed as explained before the example. For example, the (normal) transition from state $(0,1)$ to state $(1,4)$ has probability $\mathscr{A}_a(4,1)$, which is equal to the probability that S took a transition from state 1 to state 4 and produced output a. Similarly, the (erroneous) transition from state $(0,2)$ to state $(0,3)$ has probability $p_{d_b} \cdot \mathscr{A}_b(3,2)$, which is the probability that b was produced by S and then it was deleted by the sensors. □

Posterior Probability Calculation

The recursive algorithm described in the preliminaries cannot be applied in the case of sensor unreliabilities (see Figure 3). The challenge here is the existence of arcs that are vertical within a stage, possibly forming loops (as in our example) which may be traversed any number of times (more loop traversals occur, of course, with lower probability). Next, we establish some notation which will help us perform the recursive calculation.

We can view the trellis diagram for a given observed sequence z_1^L as a probabilistic FSM H with $|Q_H| = (L+1) \cdot |Q|$ states and probabilities on transitions determined by the trellis diagram. The transition probabilities do not generally satisfy the Markovian property and the matrix $\mathscr{A}_H$ that describes the transition probabilities of FSM H is not stochastic. We can easily build H' by modifying H, so that the assigned transition probabilities produce a Markov chain and, in particular, an absorbing Markov chain. More specifically, we append $|Q|+1$ states following two steps:

1. We add an extra stage at the end of the trellis diagram, i.e., we add $|Q|$ states of the form $(L+1, j)$ so that from each state of the form (L, j) there exists a transition with probability one to state $(L+1, j)$, $j \in \{1, 2, ..., |Q|\}$; we also add a self-loop with probability one at each state of the form $(L+1, j)$. We call each state of the form $(L+1, j)$ a *consistent* state because H' being in that state implies that S is consistent with the observed sequence z_1^L.
2. We add state q_{in} to represent the *inconsistent* state, i.e., H is in state q_{in} when the observed sequence is not consistent with S. To achieve this, we add transitions from each state of FSM H to the inconsistent state q_{in} with probability such that the sum of the transition probabilities leaving each state is equal to one; we also add a self-loop at state q_{in} with probability one.

The resulting Markov chain H' has $|Q_{H'}| = (L+2) \cdot |Q| + 1$ states. The only self-loops in H' with probability one are those in the consistent states (of the form $(L+1, j)$) and in the inconsistent state (q_{in}). In fact, due to the particular structure of H' (and given that there is a nonzero probability to leave the vertical loop at each stage), the consistent and inconsistent states are the only absorbing states, while the rest of the states are transient. Therefore, when H' reaches its stationary distribution, only the absorbing states will have nonzero probabilities (summing up to one). We are interested in the stationary distribution of H' so that we can account for output sequences $y_1^{L_y}$ of any length that correspond to the observed sequence z_1^L, i.e., for $L_y = L, L+1, ..., \infty$. (Recall that without sensor failures we have $L_y = L$.)

More formally, we arrange the states of H' in the order $(0,1)$, $(0,2), ..., (0,|Q|), (1,1), (1,2)..., (1,|Q|), ..., (L+$

$1,1),(L+1,2),...,(L+1,|Q|),q_{in}$. Let $\pi_{H'}[0]$ be a vector with $|Q_{H'}|$ entries, each of which represents the initial probability of each state of H'. We are interested in the stationary probability distribution of H' denoted by

$$\pi_{H'} = \lim_{n\to\infty} \pi_{H'}[n] = \lim_{n\to\infty} \mathscr{A}_{H'}^n \cdot \pi_{H'}[0],$$

where the state transition matrix $\mathscr{A}_{H'}$ of H' is in its canonical form given by

$$\mathscr{A}_{H'} = \begin{bmatrix} \mathscr{A}_H & \mathbf{0} \\ R & \mathbf{I} \end{bmatrix}. \quad (1)$$

Here $\mathscr{A}_H$ captures the behavior of the transient states of H', the $(|Q|+1)\times|Q_H|$ matrix R captures the transitions from the transient states to the absorbing states, $\mathbf{0}$ is a matrix of appropriate dimensions with all zero entries, and $\mathbf{I}$ is the identity matrix of appropriate dimensions. Note that since H' is an absorbing Markov chain, the limit $\lim_{n\to\infty}\mathscr{A}_{H'}^n$ exists and it is given by

$$\lim_{n\to\infty} \mathscr{A}_{H'}^n = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ (\mathbf{I}-\mathscr{A}_H)^{-1}R & \mathbf{I} \end{bmatrix}, \quad (2)$$

where $(\mathbf{I}-\mathscr{A}_H)^{-1}$ is called the fundamental matrix (Kemeny, Snell, & Knapp 1976).

The only nonzero entries of $\pi_{H'}$ are those that correspond to the consistent and inconsistent states. In fact, the probability that H' ends up in a consistent state is equal to the complement of the probability that H' ends up in the inconsistent state and it is equal to the probability of the observed sequence z_1^L given the FSM model S, i.e.,

$$P(z_1^L \mid S) = \sum_{j=L\cdot|Q|}^{|Q_H|-1} \pi_{H'}(j) = 1 - \pi_{H'}(|Q_{H'}|).$$

Note that the above result for the posterior probability calculation is consistent with the one obtained in (Athanasopoulou, Li, & Hadjicostis 2006) using the notion of the observation FSM and its composition with S.

Recursive Posterior Probability Calculation

In this section we exploit the structure of matrix $\mathscr{A}_{H'}$ which captures the transition probabilities of H' to perform the posterior probability calculations in an efficient manner. We first define the following submatrices which will be used to express $\mathscr{A}_{H'}$.

- Matrices $B_{m,m+1}$, $m = 0,1,...,L$, capture the transitions from any state of H' at stage m to any state of H' at stage $m+1$. They can be obtained from $\mathscr{A}_{H'}$ as $B_{m,m+1}(k,j) = \mathscr{A}_{H'}((m+1)\cdot|Q|+k, m\cdot|Q|+j)$, where $k,j = 1,2,...,|Q|$.
- Matrices B_m, $m = 0,1,...,L$, capture the vertical transitions and account for deletions. They can be obtained from $\mathscr{A}_{H'}$ as $B_m(k,j) = \mathscr{A}_{H'}(m\cdot|Q|+k, m\cdot|Q|+j)$, where $k,j = 1,2,...,|Q|$. (Note that if deletions occur at each observation step with the same probability, then $B_m = B$, $m = 0,1,...,L$.)
- C^T is a row vector with entries $C^T(j) = 1 - \sum_{k=1}^{|Q_H|}\mathscr{A}_H(k,j)$, for $j = 1,2,...,|Q_H|$, i.e., C^T ensures that the sum of each column of $\mathscr{A}_{H'}$ is equal to 1.

We should note here that an alternate way to compute the block matrices $B_{m,m+1}$ and B_m directly, without the help of the modified trellis diagram, is by using the following equations:

$$B_m(k,j) = \sum_{\substack{\forall d_{\sigma'}\in D \text{ s. t.} \\ \delta(j,out(d_{\sigma'}))=k}} (p_{d_{\sigma'}}[m]\cdot\mathscr{A}_{\sigma'}(k,j)),$$

$$B_{m,m+1}(k,j) = (1 - \sum_{\substack{\forall d_{\sigma'}\in D \text{ s. t.} \\ \delta(j,out(d_{\sigma'}))\neq\emptyset}} p_{d_{\sigma'}}[m])\cdot\mathscr{A}_{z[m+1]}(k,j),$$

where $k,j = 1,2,...,|Q|$ and $p_{d_{\sigma'}}[m]$, $\mathscr{A}_\sigma$ were defined earlier.

Using the above notation, we can decompose the matrix $\mathscr{A}_{H'}$ in blocks and express it as

$$\mathscr{A}_{H'} = \left[\begin{array}{cccccc|cc} B_0 & 0 & 0 & ... & 0 & 0 & 0 & 0 \\ B_{0,1} & B_1 & 0 & ... & 0 & 0 & 0 & 0 \\ 0 & B_{1,2} & B_2 & ... & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & ... & B_{L-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & ... & B_{L-1,L} & B_L & 0 & 0 \\ \hline 0 & 0 & 0 & ... & 0 & \mathbf{I}-B_L & \mathbf{I} & 0 \\ & & & C^T & & & 0 & 1 \end{array}\right].$$

Recall that the matrix $\mathscr{A}_{H'}$ is in its canonical form (see Equation 1), where the submatrices $\mathscr{A}_H$ and R are given by

$$\mathscr{A}_H = \begin{bmatrix} B_0 & 0 & 0 & ... & 0 & 0 \\ B_{0,1} & B_1 & 0 & ... & 0 & 0 \\ 0 & B_{1,2} & B_2 & ... & 0 & 0 \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots \\ 0 & 0 & 0 & ... & B_{L-1} & 0 \\ 0 & 0 & 0 & ... & B_{L-1,L} & B_L \end{bmatrix},$$

$$R = \begin{bmatrix} 0 & ... & 0 & \mathbf{I}-B_L \\ & & C^T & \end{bmatrix}.$$

In the initial probability distribution vector $\pi_{H'}[0]$ the only nonzero entries are its first $|Q|$ entries, i.e., $\pi_H[0] = (\rho[0]\ 0\ ...\ 0)^T$, where $\rho[0]$ denotes the initial probability distribution of S (i.e., it is a $|Q|$-dimensional vector, whose j^{th} entry denotes the probability that S is initially in state j). Recall that $\pi_{H'}$ denotes the stationary probability distribution vector of H' and has nonzero entries only in the absorbing states, i.e., its last $|Q|+1$ states. Hence, given the observed sequence z_1^L, we can express $\pi_{H'}$ as $\pi_{H'} = (0\ ...\ 0\ \rho[\mathrm{L}{+}1]\ p_{in}[L+1])^T$, where $\rho[L+1]$ captures the probabilities of the consistent states and $p_{in}[L+1]$ denotes the probability of the inconsistent state. The following equations hold:

$$\begin{aligned} \pi_{H'} &= \lim_{n\to\infty}\mathscr{A}_{H'}^n\cdot\pi_{H'}[0] \\ (0\ ...\ 0\ \rho[L+1]\ p_{in}[L+1])^T &= \lim_{n\to\infty}\mathscr{A}_{H'}^n\cdot(\rho[0]\ ...\ 0)^T \\ \rho[L+1] &= \lim_{n\to\infty}\mathscr{A}_{H'}^n(L+2,1)\cdot\rho[0]. \end{aligned}$$

Therefore, in order to calculate the probability of the consistent states we only need the initial distribution of S and the $(L+2,1)^{th}$ element of the matrix $\lim_{n\to\infty}\mathscr{A}_{H'}^n$.

Next, we argue that we can compute the $\lim_{n\to\infty} \mathscr{A}_H^n(L+2,1)$ with much less complexity than the standard computation in Equation 2. For simplicity, we focus on the case where the observed sequence is of length 2, i.e., $L=2$. The state transition matrix $\mathscr{A}_{H'}$ is given by

$$\mathscr{A}_{H'} = \left[\begin{array}{ccc|cc} B_0 & 0 & 0 & 0 & 0 \\ B_{0,1} & B_1 & 0 & 0 & 0 \\ 0 & B_{1,2} & B_2 & 0 & 0 \\ \hline 0 & 0 & \mathbf{I}-B_2 & \mathbf{I} & 0 \\ & C^T & & 0 & 1 \end{array}\right].$$

We can compute by induction the matrix $\mathscr{A}_{H'}$ raised to the power n and consequently find the limit of $\mathscr{A}_{H'}^n,(4,1)$ as n tends to infinity as follows.

$$\begin{array}{l} \lim_{n\to\infty} \mathscr{A}_H^n(4,1) = \\ \lim_{n\to\infty} \sum_{j_1+j_2+j_3+j_4=n-3} \mathbf{I}^{j_4}\, \mathbf{I}\, B_2^{j_3} B_{1,2} B_1^{j_2} B_{0,1} B_0^{j_1} = \\ (\mathbf{I}+B_2+B_2^2+...)B_{1,2}(\mathbf{I}+B_1+B_1^2+...) \\ B_{0,1}(\mathbf{I}+B_0+B_0^2+...) = \\ \left(\sum_{j=0}^{\infty} B_2^j\right) B_{1,2} \left(\sum_{j=0}^{\infty} B_1^j\right) B_{0,1} \left(\sum_{j=0}^{\infty} B_0^j\right) = \\ (\mathbf{I}-B_2)^{-1}B_{1,2}(\mathbf{I}-B_1)^{-1}B_{0,1}(\mathbf{I}-B_0)^{-1}. \end{array}$$

(The detailed proof is omitted due to space limitations.)

As explained earlier, we are interested in the state probabilities of the *consistent* states, which will be given by the entries of the vector $\rho[3] = \lim_{n\to\infty} \mathscr{A}_H^n(4,1)\,\rho[0]$. From the above equation, we get

$$\rho[3] = ((\mathbf{I}-B_2)^{-1}B_{1,2}(\mathbf{I}-B_1)^{-1}B_{0,1}(\mathbf{I}-B_0)^{-1})\rho[0].$$

To simplify notation let us define $B'_{m,m+1} = (\mathbf{I}-B_{m+1})^{-1}B_{m,m+1}$, $m=0,1,2$. Hence, $\rho[3] = B'_{1,2}B'_{0,1}(\mathbf{I}-B_0)^{-1}\rho[0]$.

Generalizing the above result for any number of observations L, the vector that describes the probabilities of the consistent states given the observed sequence z_1^L satisfies

$$\rho[L+1] = \left(\prod_{i=0}^{L-1} B'_{m,m+1}\right)(\mathbf{I}-B_0)^{-1}\,\rho[0], \tag{3}$$

where $B'_{m,m+1} = (\mathbf{I}-B_{m+1})^{-1}B_{m,m+1}$, $m=0,1,...,L$.

By inspection of Equation 3 we notice that the computation of $\rho[L+1]$ can be performed recursively as follows:

$$\begin{array}{l} \rho[1] = B'_{0,1}(\mathbf{I}-B_0)^{-1}\,\rho[0], \\ \rho[m+1] = B'_{m,m+1}\,\rho[m],\ m=1,2,...,L, \end{array} \tag{4}$$

where $\rho[m+1]$ represents the probability of consistent states given the observed sequence z_1^m. The probability that the observed sequence z_1^L was produced by the particular FSM S is equal to the sum of the elements of the state probability distribution vector $\rho[L+1]$, i.e., $P(z_1^L \mid S) = \sum_{j=1}^{|Q|}\rho[L+1](j)$.

The above algorithm is described in pseudocode as follows.

Algorithm Input: Matrices $\{B_m\}$, $\{B_{m,m+1}\}$, where $m=0,1,\ldots,L$; an observed (possibly corrupted) output sequence $z_1^L=\{z[m]\}$ where $m=1,\ldots,L$, and the initial probability distribution $\rho[0]$.

1. Initialization. Let $m=0$, $z[m]=\emptyset$,
   compute $B'_{0,1} = (\mathbf{I}-B_1)^{-1}B_{0,1}$,
   compute $\rho[1] = B'_{0,1}(\mathbf{I}-B_0)^{-1}\,\rho[0]$.
2. Let $m=1$.
3. Consider the output $z[m]$, do
   compute $B'_{m,m+1} = (\mathbf{I}-B_{m+1})^{-1}B_{m,m+1}$,
   compute $\rho[m+1] = B'_{m,m+1}\rho[m]$.
4. $m=m+1$.
5. If $m=L+1$, Goto 6; else Goto 3.
6. Compute $P(z_1^L \mid S) = \sum_{j=1}^{|Q|}\rho[L+1](j)$. □

To gain some intuition regarding the recursion, let us consider for now the case of reliable sensors. This case corresponds to matrices B_m in $\mathscr{A}_{H'}$ being equal to zero, which means that there are no vertical transitions (transitions within the same stage) in the trellis diagram. The recursion (Equation 4) becomes $\rho[m+1] = B_{m,m+1}\cdot\,\rho[m]$, $m=0,1,...,L$. The latter equation is the same as the recursive equation that appeared in the preliminaries for the case of reliable sensors (where we denoted $B_{m,m+1}$ by $\mathscr{A}_{y[m+1]}$). Intuitively, every time we get a new observation we update the current probability vector by multiplying it with the state transition matrix of S that corresponds to the new observation. With the above intuition at hand, we now return to the case of sensor failures. Here, we also need to take into consideration the fact that any number of vertical transitions may occur. Therefore, every time we get a new observation $z[m+1]$, we multiply the current probability vector with the state transition matrix of S that corresponds to the new observation (as before) and also with $(\mathbf{I}-B_{m+1})^{-1} = \sum_{j=0}^{\infty} B_{m+1}^j$ thereby taking into account the vertical transitions at stage $m+1$.

The matrices $B_{m,m+1}$ have dimension $|Q|\times|Q|$, while the matrix $\mathscr{A}_H$ has dimension $|Q_H|\times|Q_H|$, where $|Q_H| = (L+2)\cdot|Q|$. If we calculate π_H without taking advantage of the structure of $\mathscr{A}_H$, the computational complexity is proportional to $O(((L+2)\cdot|Q|)^3) = O(L^3\cdot|Q|^3)$. If we use the recursive approach instead, the computational complexity reduces significantly to $O((L+2)\cdot(|Q|^2+|Q|^3)) = O(L\cdot|Q|^3)$ (each stage requires the inversion of a new $|Q|\times|Q|$ matrix which has complexity $O(|Q|^3)$ and dominates the computational complexity associated with that particular stage). If sensor failure probabilities remain invariant at each stage, then matrix B_m at stage m only needs to be inverted once and the complexity of the recursive approach is $O((L+2)\cdot|Q|^2+|Q|^3) = O(L\cdot|Q|^2+|Q|^3)$. In addition to complexity gains, the recursive nature of the calculations allows us to monitor the system under diagnosis online and calculate the probability of the observed sequence at each observation step by first updating the state probability vector and then summing up its entries.

A Failure Diagnosis Example

As an application of our techniques, we consider the logical link control sublayer in the IEEE/Std 802.2 local area net-

Table 1: State transition functionality of FSM shown in Figure 4. (Each entry in the table shows the inputs that take the FSM from the current state (corresponding to the entry's row) to the next state (corresponding to the entry's column).)

Current State \ Next State	ADM	RESET	ERROR	D_CONN	SETUP	NORMAL
ADM					Con_Req	R_Sabm
RESET	R_Disc_Cmd	R_Sabm				R_Ua_Rsp
ERROR	R_Dm_Rsp, R_Disc_Cmd	TExp>N2	TExp, R_Ua_Rsp	R_Frmr		R_Sabm
D_CONN	R_Dm_Rsp, R_Sabm, R_Ua_Rsp, TExp>N2			R_Disc_Cmd		
SETUP	TExp>N2				R_Sabm, TExp	R_Ua_Rsp
NORMAL			R_Ua_Rsp, R_Invi_Cmd	R_Frmr_Rsp, R_Disc_Req		R_Sabm

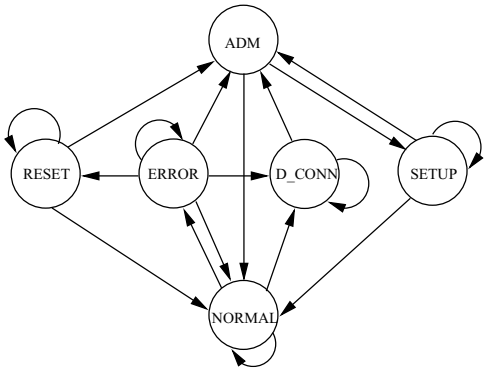


Figure 4: FSM model for part of 802.2 protocol responsible for data link establishment, disconnection, and resetting states.

work protocol (Std 1985), which is a peer protocol for use in a multi-station multi-access environment. More specifically, we consider the part of the protocol which is responsible for data link establishment, disconnection, and link resetting, and which is modeled as the six-state FSM shown in Figure 4 with state transition functionality as defined in Table 1.

The system to be diagnosed in our example is a system that supposedly complies to the 802.2 standard. The model of the protocol and a model of a faulty (bogus) implementation of the protocol are known *a priori*. In order to formulate our probabilistic framework we assume that the probability distribution of the inputs is known (e.g., these probabilities have been obtained from empirical measurements). In order to keep the example here simple (and without any loss in generality), we associate three outputs to the inputs, i.e., the output set is $Y = \{a, b, c\}$ so that the resulting FSM denoted by S_1 has six states and three outputs as shown in Figure 5. We assume that input x_1 appears with probability $\frac{1}{3}$ and the remaining four inputs (namely x_2, x_3, x_4, x_5) appear with probability $\frac{1}{6}$ each.

The particular fault we are interested in is a faulty transition from state "NORMAL" to state "ERROR" instead of state "D_CONN" under the transition with output a; this could be, for example, the result of a hardware fault or a design error or a software bug. This, together with the nominal (fault-free) model description in Figure 4, fully describe the faulty model denoted by S_2. Our goal is to determine whether the underlying FSM is executing S_1 or S_2 when the observed sequence is $z_1^4 =< bacd >$. The additional challenge is that the output sequence can be corrupted due to sensor failures. For this example, we consider that output a may be deleted with probability $p_{d_a} = 0.15$.
We follow the recursive approach to compute the probability that the given observed sequence is produced by S_1 and the probability that it is produced by S_2, as shown in the following tables.

m	$\rho_1^T[m]$	$\Sigma_{j=1}^6 \rho_1[m](j)$
0	[1/6 1/6 1/6 1/6 1/6 1/6]	1
1	[0.0861 0.0278 0 0.0015 0.0086 0.0278]	0.1518
2	[0.0227 0 0 0.0108 0.0596 0]	0.0931
3	[0.0015 0 0 0.0004 0.0200 0.0076]	0.0295
4	[0.0001 0 0.0025 0.0001 0 0]	0.0027

m	$\rho_2^T[m]$	$\Sigma_{j=1}^6 \rho_2[m](j)$
0	[1/6 1/6 1/6 1/6 1/6 1/6]	1
1	[0.2323 0 0 0.1502 0.1343 0]	0.5168
2	[0.0794 0 0 0.0812 0.1628 0]	0.3234
3	[0.0860 0 0 0.0439 0.0615 0]	0.1914
4	[0.0353 0 0 0.0237 0.0609 0]	0.1199

As long as the prior probabilities for the two models satisfy $\frac{P_1}{P_2} < \frac{0.1199}{0.0027} = 44.4074$, the MAP rule implies that, to minimize the probability of error, we will decide that the underlying implementation of the protocol is a faulty one.

Summary and Discussion

In this paper we proposed a recursive methodology to calculate the probability of an observed, possibly erroneous sequence given two finite state models (one of which can be thought of as fault-free and the other one as faulty). Our

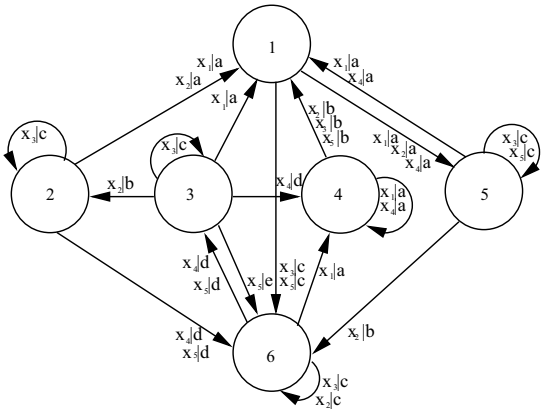


Figure 5: State transition diagram of FSM S_1.

goal is to determine which model has more likely produced the observed sequence. We considered sensor failures which can corrupt the observed sequence and we assumed that these errors are (conditionally) independent from the inputs. By constructing the trellis diagram which includes all possible sequences consistent with both the observations and the given FSMs, we developed a recursive algorithm that efficiently computes the total probability with which each FSM model, together with a combination of sensor failures, can generate the observed sequence. Our algorithm can deal with vertical loops in the trellis diagrams which can be caused by output deletions.

In this work, we considered the fault-free operation of the system and one mode of a faulty operation (given information from unreliable sensors). Multiple faulty operation modes can be handled in a straightforward manner by our proposed algorithm (by evaluating how well the observations match each possible model so that we can eventually choose the best match). However, following our current approach, we would need to invoke the algorithm as many times as the number of operation modes of the system. We plan to extend our current approach to situations where we can take advantage of the system structure to evaluate the likelihood of multiple faulty models more efficiently. One possible extension is to introduce more structured fault-free model as well as more structured faulty models. For example, we could consider systems that consist of some independent or loosely coupled components. Another direction would be to use factorial hidden Markov models by imposing constraints on the state transition functionality of the system so that each state variable evolves independently of the remaining variables and it is *a priori* decoupled from the others. Our results can be easily extended to classification of several hidden Markov models with applications in various fields such as document or image classification, pattern recognition, and bioinformatics.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation under NSF Career Award No 0092696 and NSF ITR Award No 0426831, and in part by the Air Force Office of Scientific Research under Award No AFOSR DoD F49620-01-1-0365URI. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NSF or AFOSR.

References

Athanasopoulou, E.; Li, L.; and Hadjicostis, C. N. 2006. Probabilistic failure diagnosis in finite state machines under unreliable observations. In *Proc. Int. Workshop on Discrete-Event Systems*.

Blanke, M.; Kinnaert, M.; Lunze, J.; and Staroswiecki, M. 2003. *Diagnosis and Fault-Tolerant Control*. Springer-Verlag.

Cassandras, C. G., and Lafortune, S. 1999. *Introduction to Discrete-Event Systems*. Kluwer.

Ephraim, Y., and Merhav, N. 2002. Hidden Markov processes. *IEEE Trans. Information Theory* 48(6):1518.

Hadjicostis, C. N. 2005. Probabilistic detection of FSM single state-transition faults based on state occupancy measurements. *IEEE Trans. on Automatic Control* 50(12):2078.

Jordan, M. 2004. Graphical models. *Statistical Sciences, Special Issue on Bayesian Statistics* 19(1):140.

Kemeny, J. G.; Snell, J. L.; and Knapp, A. W. 1976. *Denumerable Markov Chains*. Springer-Verlag.

Lin, F. 1994. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems: Theory and Applications* 4(2):197.

Poritz, A. M. 1988. Hidden Markov models: A guided tour. *Proc. IEEE Conf. Acoustics, Speech, and Signal Processing* 1:7.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proc. IEEE*, volume 77.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Trans. on Automatic Control* 40(9):1555.

1985. American National Standards Iinstitute, ANSI/IEEE Std. 802.2, The Institute of Electrical and Electronics Engineers, Logical link control.

Thorsley, D., and Teneketzis, D. 2005. Diagnosability of stochastic discrete-event systems. *IEEE Trans. on Automatic Control* 50(4):476.

Vidal, E.; Thollard, F.; de la Higuera, C.; Casacuberta, F.; and Carrasco, R. C. 2005. Probabilistic finite-state machines–Part I. *IEEE Trans. Pattern Anal. and Machine Intell.* 27(7):1013.

Wu, Y., and Hadjicostis, C. N. 2005. Algebraic approaches for fault identification in discrete-event systems. *IEEE Trans. on Automatic Control* 50(12):2048.

Zad, S. H.; Kwong, R. H.; and Wonham, W. M. 2003. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Trans. on Automatic Control* 48(7):1199.

A Diagnosis Driven Self-Reconfigurable Filter

Emmanuel Benazera
Robotics Group
Bremen Universität
Robert-Hooke-Str. 5,
D-28359, Bremen, Germany
benazera@informatik.uni-bremen.de

Louise Travé-Massuyès
LAAS-CNRS
7, av. du Colonel Roche
31077 Toulouse Cedex4, France
louise@laas.fr

Abstract

Filtering consists in estimating the value of system state variables based on available noisy measurements. In Artificial Intelligence (AI), reasoning from first principles uses logic to trace back influences among variables and finds minimal sets that can be held responsible for a given measurement. Both theories rely on a model of the system, but while filtering implements an error feedback mechanism that closes on the measurements, reasoning from first principles provides the ability to localise the causes from the effects. In certain cases, when a system misbehaves, e.g. the motor in a robotic arm joint starts failing, the filter is able to detect the drift, but unable to locate the problem with precision in the state-space. The ability to break up the filter's feedback loop in such cases is exactly the purpose of our approach. We aim at coupling the localization ability of the theory of diagnosis from first principles with the state estimation achievement of Kalman filtering. The targeted result is a novel filter which localizes the subpart of the system that is misbehaving, isolates its effects, and keeps tracking a partial state.

Introduction

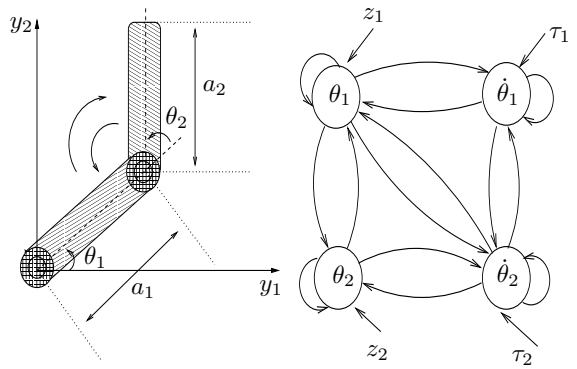
There exist numerous strategies for tracking the state of a possibly faulty system, using noisy measurements. The implied stochasticity of the system dynamics together with the number of faulty situations to account for makes it necessary to track a high number of behavioral hypotheses simultaneously. This is typically done by running either a bank of filters or a cloud of particles (Doucet *et al.* 2000). In most cases, the number of trajectories is untractable, or it is simply counter-productive to track them since many states are in fact never reached. For this reason, research has concentrated on ways to drive the filter's focus on the subset of relevant hypotheses (Hofbaur & Williams 2002a; Narasimhan, Dearden, & Bénazéra 2004) and to mitigate the blowup in tracked states (Hutter & Dearden 2003; Bénazéra & Travé-Massuyès 2003).

While these strategies are effective in practice, not all hypotheses can be modeled, of course, and more so in the case of fault hypotheses, whose number is potentially infinite. An alternative is to design a filter that tracks the potentially unmodeled behaviors. This can be done by fitting parameters to a skeleton model, e.g. using Generalized Likelihood Ratio or Expectation Maximization (Basseville & Nikiforov 1992). The problem is then to anticipate appropriate skeleton models.

In this paper, we adopt a different point of view on the problem of tracking the state of a system. Our approach is based on a reference behavior model (e.g. that of nominal behavior) but instead of closing on all the measurements, we propose to scale the filter so that it only closes on the part of the system that can be trusted to correspond to the reference model. It is necessary that such a filter correctly identifies the variables that fit the model, leaving the others in open loop. The filter naturally leaves the uncertainty to grow on these latter variables. The rational behind it is that the system upper controlling layers act locally on the estimated uncertainty, or *level of unknowingness*, instead of aiming at identifying a fully fitted model. Building the filtering loop to this end is challenging.

First, the subpart of the system and corresponding subset of variables whose behavior does not fit the reference model have to be identified. Although the numerical feedback loop that is natural to most filters makes it difficult to isolate these variables, we argue that they can actually be determined through causal analysis by logically tracing causes from their effects in the causal structure of the model. In AI, a logical theory of diagnosis does exist that can just do that. Diagnosis from first principles (DX) logically infers the minimal sets of elementary components that can be held responsible for a discrepancy in the system (Hamscher, Console, & J. de Kleer 1992). We use the power of this inference to break up the filter feedback loop after projecting the component sets on the corresponding sets of *untrusted* variables. Untrusted variables are hence decoupled from the filter loop. Second, the estimation step needs to be revised so that effects of untrusted variables are prevented from affecting themselves and *sane* variables, while discrepant measurements must not be used for updating the filter's innovation.

The aim of this paper is to bring a reasoning layer as well as a partial covariance minimization scheme into existing filters, starting with the Unscented Kalman filter that applies to nonlinear systems. The contribution stands on the idea of coupling a filtering technique well-known in the Control diagnosis community (FDI) with logical diagnosis inference from the AI diagnosis community (DX). It hence fits into the BRIDGE framework aiming at creating synergies between



(a) Support to the dynamic model.

(b) Causal System Description (CSD) for the robotic arm.

Figure 1: Two-link planar arm representation.

the FDI and DX communities (Gautam *et al.* 2004).

The paper is organised as follows. The next section presents our case study, which is a planar arm with two joints. The succeeding section overviews the principles of model based diagnosis and presents how causal models can be used. This is then interpreted in matricial form, bringing it back to the same framework as filtering methods, and the computation methods for deriving conflicts and diagnoses in this framework are presented. Following is the presentation of the Unscented Kalman Filter and how it can be modifed for partial state hypothesis filtering. Finally our semi-closed loop filter SCL-UKF that accounts for logical diagnosis inference is provided. Early results of the application of SCL-UKF to the planar arm are given and discussed. The paper ends with a section discussing related and future works.

Case study

A two-link arm example

Our case study is a two-link planar arm with two joints, at the shoulder and at the elbow. The state of the system is represented by a vector $x = (\ \theta_1\ \theta_2\ \dot{\theta}_1\ \dot{\theta}_2\ )$ where θ_1,θ_2 are the angular positions of the shoulder and elbow joints, respectively. The angular positions θ_1 and θ_2 are measured. m_1, m_2 are the respective masses of each link. Figure 1(a) pictures a schematic support to the arm dynamic model of figure 2. Our model of the arm includes a PD controller, which allows for the two angular position inputs to be translated into the input torques τ_1 and τ_2. While the model is simple enough, the number of possible faults is staggering. Component-wise, both joints can fail, the mass of the second limb can vary when used to pick up objects. Sensors and the controller may also fail. State-wise, this corresponds to 4 single discrepancies of angular positions and speeds, which yield 2^4 multiple faults, 2^6 with sensor faults, and 2^{10} with controller faults. Thus for such a small system, an exhaustive multi-hypothesis filter would require 2^{10} hypotheses to be modelled. In the following, we show how to build a single filter that does reconfigure itself instead of relying on hypotheses to be modelled.

Diagnosis from First Principles

Diagnosis oriented causal modelling

Reasoning about non-linear systems can be supported by a causal representation of influences among variables. Influences are a conceptualisation of the links established by the components between variables in a system. In fact, causal models have been proposed and shown to be suitable for diagnosis in several pieces of work (Biswas & Manders 2006; Travé-Massuyès *et al.* 2001; Travé-Massuyès & Calderón-Espinoza 2007). The model causal structure then acts as a substitute of dependency recording mechanisms. Causal models are generally supported by an oriented graph, also called *Causal Graph*, in which nodes represent variables and edges represent influences from variable to variable. An oriented edge from variable v_i to variable v_j exists if v_i has an influence on v_j, i.e. if a perturbation on variable v_i affects the value of variable v_j. v_i and v_j are called the *cause* and the *effect* variable of the influence, respectively. Three types of variables exist to model a system:

- *Input variables* are exogenous to the system. Their values are controlled by the system's environment and assumed to be known. n_u is the number of input variables.
- *Measured or output variables* are known, as provided by a sensoring device. n_z is the number of measured variables.
- *State variables* are internal to the model and their values are not known. n_x is the number of internal variables.

Definition 1 (Causal System Description (CSD))**.** *Let $CSD = \{V, \mathcal{I}\}$ be the causal system description where V is the set of variables that define the system, and $\mathcal{I}$ the set of oriented influences that model dependencies.*

Conflicts and Diagnoses

Let's assume that a fault detection mechanism is available and that it activates an alarm when the measured value (also called *observation*) of an output variable is not consistent with the expected value. Such a discrepancy for a measured variable z eventually indicates a misbehavior.

Definition 2 (Discrepant output vector)**.** *Let Z be the vector of output variables. The discrepant observation vector Z^f is a vector of size n_z such that* $z_i^f = \begin{cases} 1 \text{ if } z_i \text{ is discrepant} \\ 0 \text{ otherwise.} \end{cases}$

When one or several output variables misbehave, we can derive all sets of faulty influences that may explain the observations. The influences that may be at the origin of the misbehavior of a variable z_i are those related to the edges belonging to the paths going from the measured nodes to the node representing z_i, also called *ascending influences*. The set of such influences is a *conflict* set in the sense of (Reiter 1987). Conflict sets are sets of influences that cannot behave normally altogether according to the observations. A minimal conflict is a conflict that does not strictly include (in the sense of set inclusion) any conflict. (Reiter 1987) proved that minimal diagnoses can be computed from minimal conflicts.

$$M(x)\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}+\begin{bmatrix}-m_2a_1a_2(2\dot{\theta}_1\dot{\theta}_2+\dot{\theta}_2^2)+\sin\theta_2\\ m_2a_1a_2\dot{\theta}_1^2\sin\theta_2\end{bmatrix}+\begin{bmatrix}(m_1+m_2)ga_1\cos\theta_1+m_2ga_2\cos\theta_1+\theta_2\\ m_2ga_2\cos\theta_1+\theta_2\end{bmatrix}=\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$$

where

$$M(x)=\begin{bmatrix}(m_1+m_2)a_1^2+m_2a_2^2+2m_2a_1a_2\cos\theta_2 & m_2a_2^2+m_2a_1a_2\cos\theta_2\\ m_2a_2^2+m_2a_1a_2\cos\theta_2 & m_2a_2^2\end{bmatrix}$$

Figure 2: Two-link planar arm dynamic model.

Proposition 1 (Minimal Diagnosis (Reiter 1987))**.** *Given a discrepant observation vector Z^f, $\Delta \subseteq \mathcal{I}$ is a (minimal) diagnosis for* (CSD, Z^f) *iff Δ is a (minimal) hitting set for the collection of (minimal) influence conflict sets.*

A hitting set of a collection of sets is a set intersecting every set of this collection.

Determining Candidate Diagnoses

In this section, we first interpret influence conflicts and diagnoses in a matricial form, suitable for coupling with the filtering framework. The computational methods for building *conflict* and *diagnosis matrices* are then presented.

Conflicts and diagnoses in a matrix framework

The causal graph associated to CSD can be equivalently represented by an incidence matrix $\mathcal{I}$, of size (n_c, n_c) with $n_c = n_x + n_u + n_z$:

$$\mathcal{I}=\begin{pmatrix}A & B & \emptyset\\ \emptyset & \mathbb{I}_u & \emptyset\\ H & \emptyset & \mathbb{I}_z\end{pmatrix}, \text{with } \mathcal{I}_{ij}=\begin{cases}1 \text{ if } x_i \text{ influences } x_j\\ 0 \text{ otherwise}\end{cases}$$

where A is of size (n_x, n_x), B of size (n_x, n_u), and H of size (n_z, n_x). These are incidence matrices that represent influences among state, input, and output variables, respectively. $\mathcal{I}$ reflects the natural hierarchy of influences: inputs on state, state on measures. $\mathbb{I}_u$ and $\mathbb{I}_z$ are identity matrices and account for effects due to external causes onto inputs (e.g. controller) and outputs (e.g. sensors).

Example. *Figure 1(b) shows the CSD=$\{V, \mathcal{I}\}$ for our case study, with $V = \begin{pmatrix}\theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 & \tau_1 & \tau_2 & z_1 & z_2\end{pmatrix}$. We have: $A = \begin{pmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{pmatrix}$, $B = \begin{pmatrix}0&0\\0&0\\1&0\\0&1\end{pmatrix}$,*

$H = \begin{pmatrix}1&0&0&0\\0&1&0&0\end{pmatrix}$ *and* $\mathcal{I} =$

$$\left(\begin{array}{cccc|cc|cc}1&1&1&1&0&0&0&0\\1&1&1&1&0&0&0&0\\1&1&1&1&1&0&0&0\\1&1&1&1&0&1&0&0\\\hline 0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\\hline 1&0&0&0&0&0&1&0\\0&1&0&0&0&0&0&1\end{array}\right)$$

For a given discrepant output vector Z^f, influence conflict sets may as well be represented in matrix form, as indicated by the following definition.

Definition 3 (Influence Conflict Matrix)**.** *Given a discrepant output vector Z^f, an influence conflict matrix Γ is an incidence matrix of size $n_c \times n_c$ whose entries correspond to ascending influences of the discrepant output variables of Z^f.*

In the above matrix, all conflicts are represented but it is difficult to identify each of them and relate them to their corresponding discrepant output variable. Now, conflicting influences naturally map onto variables and conversely. Indeed, influence conflict sets correspond to paths in the causal graph and a path may as well be represented by the edges (influences) or by the nodes (variables). This leads to the following definition.

Definition 4 (Variable Conflict Matrix)**.** *Given a discrepant output vector Z^f, a variable conflict matrix Λ is a boolean matrix of size $n_z \times n_c$ such that* $\begin{cases}\sum_j \Lambda_{i,j} > 0 \text{ if } z_i^f = 1\\ \Lambda_{i,\cdot} = 0, \text{ otherwise.}\end{cases}$

Considering a single row Λ_i of Λ we know that all state, input and output variables indicated by a non zero entry in Λ_i influence the discrepant output z_i^f. *This implies that at least one of these variables has to suffer a faulty influence to cause the discrepancy on z_i^f. Hence this set of variables can equivalently represent the influence conflict.* By *suffer a faulty influence* we mean that in the physical system, there must exist at least one influence on this variable whose effect on the discrepant output is incorrectly captured by the reference model. This set of variables is called a *variable conflict* set.A minimal variable conflict matrix is a matrix whose variable sets indicated by 1-valued entries on each row do not strictly include (in the sense of set inclusion) any variable conflict. Therefore a minimal conflict matrix indicates minimal variable conflicts only. Finally, we define the diagnosis matrix as follows.

Definition 5 (Diagnosis matrix)**.** *Given a discrepant measurement vector Z^f, a diagnosis matrix Δ is an influence incidence matrix of size $n_c \times n_c$ in which at least one faulty influence represented by a 1-value entry accounts for each discrepant measure of Z^f.*

Example. *Consider the arm's shoulder joint measure is discrepant, so $Z_f = (1 \quad 0)$.* $\Lambda = \begin{pmatrix}1&1&1&1&1&1&1&0\\0&0&0&0&0&0&0&0\end{pmatrix}$,

$$\Gamma=\left(\begin{array}{cccc|cc|cc}1&1&1&1&0&0&0&0\\1&1&1&1&0&0&0&0\\1&1&1&1&1&0&0&0\\1&1&1&1&0&1&0&0\\\hline 0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\\hline 1&0&0&0&0&0&1&0\\0&1&0&0&0&0&0&0\end{array}\right)$$

are the conflict matrices of variables and influences, respectively. Δ *with all entries equal to 0 but* $\Delta_{1,1} = 1$ *is a possible diagnosis matrix.*

Computing Conflict Matrices

The discrepant output vector leads to the identification of the matrix of conflicting influences. This section is concerned with the computational methods for building the conflict and diagnosis matrices defined above.

We suppose a discrepant output vector Z^f. H^f (of size $n_z \times n_x$) is obtained by selecting the rows of H that correspond to positive values of Z^f and zeroing the others. H^f tells which state variables directly affect the discrepant outputs. Effects of state variables on other state variables are taken into account by the matrix A. Thus we dub $X^{f,1} = H^f A$ the *discrepant state influence matrix*. In other words, variable x_i influences output z_j iff $X^{f,1}_{j,i} \neq 0$. However, $X^{f,1}$ expresses direct influences of the state on the outputs. Upstream influences can be captured iterating on A, i.e. by $X^{f,2} = X^{f,1}A$. And so on for k steps, $X^{f,k} = H^f A^k$, until $(A)^{k+1} = (A)^k$. State variable conflicts are made of all influences from state variables onto outputs, thus

$$X^f = H^f + \sum_{i=1}^{k} H^f (A)^k \tag{1}$$

Here k is such that $(A)^{k+1} = (A)^k$. We define the input influence matrix $B^f = X^f B$ where $B^f_{j,i} \neq 0$ implies that input u_i influences output z_j. Finally the matrix I^f (obtained from the identity matrix of size n_z by keeping the ones corresponding to Z^f) is used to account for sensor failures.

Example. *As before, consider* $Z^f = (1 \quad 0)$. *Therefore* $H^f = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{smallmatrix}\right)$. $A^2 = \mathbb{I}_x$, *and* $X^f = \left(\begin{smallmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{smallmatrix}\right)$, $B^f = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix}\right)$, $\mathbb{I}^f = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$.

Now, we can build the variable conflict matrix Λ as the concatenation of matrices (X_f, B_f, I_f). Following the consistency-based theory presented above, Λ is the conflict matrix because each of its rows indicates an influence *conflict*.

Example. *Following up on our example:*

$$\Lambda = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Algorithm 1 sums up the steps of the automated generation of Λ.

1: Build H_f from the discrepant lines of H.
2: Compute powers of A.
3: Compute X^f.
4: Compute B^f.
5: Build $\Lambda \leftarrow (X^f, B^f, I^f)$.

Algorithm 1: Conflict generation.

Proposition 2 (Minimal Conflict matrix)**.** *Given a discrepant output vector* Z^f, Λ *is the minimal conflict matrix w.r.t.* Z^f.

Proof. The conflict matrix of variables Λ contains all variables that can held responsible for a discrepant variable. In a graph theoretic framework, the matrix of conflicting influences Γ contains all edges that belong to paths from input, state and output vertices to the discrepant vertices. Paths of increasing lengths correspond to the powers 1 to k of the incidence matrix A. Considering a path p_i in this graph, and assuming that one influence I_r is removed, leads to a subpath sp_i. Then sp_i is no conflict since if I_r is faulty, all the influences in sp_i can be normal, the discrepancy being hence explained by I_r only. The same applies to any subpath of p_i, meaning that p_i corresponds to a minimal conflict. □

Computing diagnosis matrices

Hitting sets based computation From the previous section it comes that the logical theory of diagnosis allows for the generation of the diagnosis candidates through the computation of the hitting sets.

Computing the diagnoses comes back to computing the hitting sets of the subset of variables indicated by each row of Λ. This computation returns the set of diagnosis matrices. An incremental algorithm to generate all the minimal hitting sets based on a set of conflicts was originally proposed by (Reiter 1987), then corrected by (Greiner, Smith, & Wilkerson 1989). This algorithm gives a means to compute diagnoses incrementally, under the permanent fault assumption. It builds a Hitting-Set tree (HS-tree) in which leaves contain the minimal diagnose. Like in (Travé-Massuyès & Calderón-Espinoza 2007), we refer to the algorithm version by (Levy 1991) which is more efficient than the original one because it uses less comparisons at each step. We implement a version of the algorithm where diagnoses are given by matrices, and where edges need not to be labelled.

Algorithm 2 begins with a tree HS consisting of a simple root, with an attached empty diagnosis matrix. Each tree node n supports a diagnosis matrix that records entries that solve the conflicts from the root node to n. The algorithm takes conflicts (vector rows of Λ) in an arbitrary order. For every conflict Λ_i and every element $\Lambda_{i,c}$ of the conflict, the algorithm builds two lists, *newleaves[c]* and *oldleaves[c]* (step 3). New leaves to a leaf l are created whenever Λ_i is not already into Δ_l. Intersection test is a matrix operation that maps influences diagnose onto conflicting variables (step 7). The conversion from state conflicts to influence conflicts is done at step 8. Step 10 creates the local diagnosis matrices, one per influence to a local conflict variable. A new leaf l is pruned if it already contains some conflicts that appear in some old leaf. At the end of the diagnosis procedure (step 19), the minimal hitting sets, and hence the minimal diagnoses that explain the system's misbehaviors, are given by the set of diagnosis matrices attached to the leaves. Note that a trivial diagnosis is one that accounts for simultaneous sensor failures.

The problem of exoneration Generating diagnoses as presented above is rather conservative since there are influences in the diagnoses that are not manifesting themselves thoroughly at the level of discrepant outputs. This occurs whenever an influence belongs to the path to several outputs

for Each conflict Λ_i in Λ (i.e. row) **do**
  for Each element $\Lambda_{i,c}$ **do**
    Initialize the lists *new-leaves*[c]={} and *old-leaves*[c]={}.
  for leaf l of HS **do**
    $\Delta^l \leftarrow$ diagnosis matrix in leaf l.
    /* creating new leaves (intersection test). */
    if $\Delta^l.\Lambda_i =$ null vector **then**
      Build Γ_i from Λ_i.
      for Each positive $\Lambda_{i,c}$ **do**
        for Each positive $\Gamma_{c,j}$ **do**
          create $\Delta \leftarrow \Delta^l$. $\Delta_{c,j} = \Gamma_{c,j}$.
          add new node (n, Δ) to l, and Δ to *new-leaves*[c].
    /* creating old leaves (intersection is singleton). */
    if $\Delta_l.\Lambda_i$ has a single positive value **then**
      add Δ_l to *old-leaves*[c].
    /* closing leaves (inclusion test). */
    for Each positive element c in Γ_i **do**
      for Each matrix Δ_n in *new-leave*[c] **do**
        if Δ_n contains some Δ_o in *old-leaves*[c] **then**
          close the branch of the node with Δ_n.

Algorithm 2: Minimal Hitting sets with diagnosis matrices.

and that not all of them are discrepant.

Example. *Given* $Z^f = \begin{pmatrix} 1 & 0 \end{pmatrix}$*, consider the reduced state diagnosis matrix* $\Delta^x = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$. $\Delta^x_{2,2}$ *corresponds to the influence of* θ_2 *on itself. It can be held responsible for the first joint discrepancy, if a component in the second joint has failed. However, the second joint's measure is not discrepant so this makes this diagnosis unlikely.*

The elimination of such cases can be dealt with by adopting the *exoneration assumption* in contrast to the *no exoneration assumption* (Cordier *et al.* 2004) :

- no exoneration assumption: the influences that lie on the path to a discrepant output are potentially identified as faulty, i.e. they belong to a conflict;
- exoneration assumption: the influences that lie on the path to a non discrepant output are assumed to be normal.

1: Given Z^f, compute conflicts Λ (Alg. 1).
2: Exoneration:
  - ascending variable matrix Λ^{ok} on non-discrepant measures (Alg. 1).
  - $\Lambda^{exo} = \Lambda \ominus \Lambda^{ok}$.
3: Compute Minimal Hitting sets on Λ^{exo}. (Alg. 2).

Algorithm 3: Computation of diagnosis matrices on exonerated conflicts

Note that the adoption of the exoneration assumption requires a thorough analysis of how the faults may manifest in a system. For instance, it may not be applicable to controlled systems in which the controller compensates for the faults or to highly non linear systems in which non linearities may hide the effect of the faults. The exoneration procedure can be efficiently implemented by removing from the conflicts the variables that affect non-discrepant outputs. This is done by generating ascending variables that influence non-discrepant outputs, i.e. gathering the variables that cannot suffer faulty influences for the outputs not to be discrepant. These variables are called *sane* variables.

Definition 6 (Matrix of sane variables)**.** *Given a discrepant output vector* Z^f*, a matrix of sane variables* Λ^{ok} *is a boolean matrix of size* $n_z \times n_c$ *such that* $\begin{cases} \sum_j \Lambda^{ok}_{i,j} > 0 \text{ if } z^f_i = 0 \\ \Lambda^{ok}_{i,.} = 0, \text{ otherwise.} \end{cases}$

The algorithm for determining Λ^{ok} is obviously the same as the conflict generation algorithm 1. The exoneration comes back to removing from the variable conflict matrix Λ all the entries that are 1 in Λ^{ok}, i.e. eliminating all the sane variables from the variable conflicts. This results in the exonerated variable conflict matrix $\Lambda^{exo} = \Lambda \ominus \Lambda^{ok}$. From there, the hitting set algorithm then performs normally on the exonerated set of conflicts. Algorithm 3 computes the diagnosis matrices on exonerated conflicts.

Partial State Hypothesis Filtering

In this section, we rely on the principles of the Uncented Kalman Filter (UKF) to build a filter that uses diagnoses to close only on those variables that can be considered unaffected by broken influences. It leaves the set of affected variables in open loop and lets the uncertainty naturally grow on these variables. This uncertainty is predicted from the model, and as such is theoretically sound. We hence derive a *semi-closed loop UKF* (SCL-UKF). This filter accurately combines the mininal state-space isolation of the previous section in open loop with a scaled a posteriori error minimization in closed loop.

Unscented Kalman filtering

Consider a discrete-time controlled process that is governed by a nonlinear stochastic difference equation (2) and a measurement equation (3).

$$x(t_i) = f(x(t_{i-1}), u(t_i), w(t_i)) \quad (2)$$
$$z(t_i) = h(x(t_i), v(t_i)) \quad (3)$$

$x(t_i)$, $u(t_i)$, and $z(t_i)$ have dimensions n_x, n_u, and n_z, respectively, and $w(t_i)$, $v(t_i)$ represent the process and measurement noise and are assumed to be independent, white and Gaussian with probability distributions $\mathcal{N}(0, Q)$, $\mathcal{N}(0, R)$ respectively. The Unscented Kalman filter (Julier & Uhlmann 1997) uses the Unscented Transform (UT) and fully captures the mean and covariance of the state vector with a minimal set of carefully choosen points, referred to as sigma points. The filter computes an unbiased estimate $\hat{x}$ of the state based on the optimal solution of the least-squares method (Kalman 1960). The state is a concatenation of the original state and noise variables $x^a = [x, w, v]$ of dimension n_a. The selection of a cloud of sigma points applies to the extended state to calculate the sigma matrix

$X^a = [X, X^w, X^v]$. Briefly, the state and error covariance are projected forward through the following equations:

$$\begin{aligned}
P^a(t_{i-1}) &= \begin{pmatrix} P(t_{i-1}) & 0 & 0 \\ 0 & P_w & 0 \\ 0 & 0 & P_v \end{pmatrix} \\
X^a(t_{i-1}) &= [\hat{x}^a(t_{i-1})\hat{x}^a(t_{i-1}) + \sqrt{(n_a+\lambda)P^a(t_{i-1})}] \\
X(t_i^-) &= f(X^a(t_{i-1}), u(t_i), X^w(t_i)) \\
\hat{x}(t_i^-) &= \textstyle\sum_{j=0}^{2n_a} W_j^m X(t_i^-) \\
P(t_i^-) &= \textstyle\sum_{j=0}^{2n_a} W_j^c [X_j(t_i^-) - \hat{x}_j(t_i^-)][X_j(t_i^-) - \hat{x}_j(t_i^-)]^T \\
Z(t_i^-) &= h(X(t_{i-1}), X^v(t_i)) \\
\hat{z}(t_i^-) &= \textstyle\sum_{j=0}^{2n_a} W_j^m Z(t_i^-)
\end{aligned}$$

where t_i^- indicates *a priori* values, and W^m and W^c are the mean and covariance sigma point weight vectors respectively. An adaptive gain factor K minimizes (in the least-square sense) the error covariance. Noisy measurements are introduced to compute the *a posteriori* state and covariance estimates. These steps summarize as:

$$\begin{aligned}
P_z(t_i) &= \textstyle\sum_{j=0}^{2n_a} W_j^c [Z_j(t_i^-) - \hat{z}_j(t_i^-)][Z_j(t_i^-) - \hat{z}_j(t_i^-)]^T \\
P_{xz}(t_i^-) &= \textstyle\sum_{j=0}^{2n_a} W_j^c [X_j(t_i^-) - \hat{x}_j(t_i^-)][Z_j(t_i^-) - \hat{z}_j(t_i^-)]^T \\
K &= P_{xz} P_z^{-1} \\
\hat{x}(t_i) &= \hat{x}(t_i^-) + K(z(t_i) - \hat{z}(t_i^-)) \\
P(t_i) &= P(t_i^-) - KP(t_i^-)K^T
\end{aligned}$$

Partial variance minimization

For a given diagnosis, we produce a partial estimate that is not subjected to the effects of faulty influences. This implies:

- not using discrepant observations and therefore cancelling the measurement noise they introduce;
- cancelling the effects of faulty influences on sane variables, i.e. not influenced by a faulty influence;
- cancelling the effects of faulty influences on the untrusted variables, i.e. influenced by a faulty influence.

The first point is achieved by reducing the output matrix to non-discrepant observable dimensions only. Second and third points lead to the cancelling in the gain computation of the error introduced by the untrusted variables. However, effects of sane variables on the untrusted variables are preserved. In the following we denote by $\breve{x}, \breve{P}, \breve{K}, \cdots$ the elements (state, covariance matrix, gain, ...) of the partial filter. So we have

$$\breve{x}(t_i) = \breve{x}(t_i^-) + \breve{K}(t_i)\Big(\breve{z}(t_i) - \breve{H}(\breve{x}(t_i^-))\Big) \quad (4)$$

where $\breve{z}$ are the non-discrepant outputs, $\breve{H}$ is the reduction of H to non-discrepant dimensions, $\breve{K}$ the gain that does not account for the error on the set of untrusted variables. It follows that the a posteriori partially estimated error $\breve{e}(t_i)$ is given by

$$\begin{aligned}
\breve{e}(t_i) &= x(t_i) - \breve{x}(t_i) \\
&= \breve{e}(t_i^-) + \breve{K}(t_i)\Big(v(t_i) - \breve{H}(\breve{e}(t_i^-) - e^f(t_i^-))\Big) \quad (5)
\end{aligned}$$

where $e^f(t_i^-)$ is an n_x dimensional vector such that $e_j^f(t_i^-) = e_j(t_i^-)$ if x_j is affected by an influence of Δ, 0 otherwise. From there, the partially updated covariance is given by[1]

$$\breve{P}(t_i) = \sum_{j=0}^{2n_a} W_j^c [(\hat{X}_j(t_i) - \hat{x}_j(t_i)) - (\hat{X}_j^f(t_i) - \hat{x}_j^f(t_i))]^T$$

with

$$\begin{cases}
\hat{X}_j(t_i) &= \hat{X}_j(t_i^-) + \breve{K}(z(t_i) - \hat{Z}_j(t_i^-)) \\
\hat{x}_j(t_i) &= \hat{x}_j(t_i^-) + \breve{K}(z(t_i) - \hat{z}_j(t_i^-)) \\
\hat{X}_j^f(t_i) &= \hat{X}_j^f(t_i^-) + \breve{K}^f(z(t_i) - \hat{Z}_j^f(t_i^-)) \\
\hat{x}_j^f(t_i) &= \hat{x}_j^f(t_i^-) + \breve{K}^f(z(t_i) - \hat{z}_j^f(t_i^-))
\end{cases}$$

with $(z(t_i) - \hat{Z}_j^f(t_i^-)) = (z(t_i) - \hat{z}_j^f(t_i^-)) = 0$ since untrusted variables are predicted, and

$$\hat{X}^f(t_i^-) = \frac{\partial F}{\partial X^f}(\hat{X}(t_{i-1})), \hat{x}^f(t_i^-) = \sum_{j=0}^{2L} W_j^m X_j^f$$

where the X_j^f are sigma points for the affected variables.[2] This leads to

$$\begin{aligned}
\breve{P}(t_i) = \breve{P}(t_i^-) &+ \breve{P}^f(t_i^-) - T^f(t_i^-) - (T^f(t_i^-))^T \\
&+ \breve{K}\breve{P}_z(t_i^-)\breve{K}^T - \breve{K}\breve{P}_{xz}^T(t_i^-) + \breve{K}\breve{P}_{xz}^f(t_i^-) \\
&- \breve{P}_{xz}(t_i^-)K^T + \breve{P}_{xz}^f(t_i^-)\breve{K}^T \quad (6)
\end{aligned}$$

where $\breve{P}^f(t_i^-) = E[e^f(t_i^-)(e^f(t_i^-))^T]$, $T^f(t_i^-) = E[\breve{e}(t_i^-)(e^f(t_i^-))^T]$ and $\breve{P}_{xz}$, $\breve{P}_{xz}^f$ are the cross-covariances. Minimizing the partial a posteriori error matrix leads to

$$\breve{K}(t_i) = (\breve{P}_{xz}(t_i^-) - \breve{P}_{xz}^f(t_i^-))\breve{P}_z^{-1} \quad (7)$$

The a posteriori update is written

$$P(t_i) = P(t_i^-) - \breve{K}\breve{P}_z(t_i^-)\breve{K}^T \quad (8)$$

Hypothesis Testing

The minimal candidate diagnoses generation procedure produces many hypotheses. Different hypotheses carry different levels of uncertainty. Observing that relation 6 rewrites

$$\begin{aligned}
\breve{P}(t_i) = \breve{P}(t_i^-) &+ \breve{P}^f(t_i^-) - T^f(t_i^-) - (T^f)^T(t_i^-) \\
&+ (\breve{P}_{xz}^f - \breve{P}_{xz})^T\breve{K}^T \quad (9)
\end{aligned}$$

and the error introduced by the untrusted state block is given by

$$P(t_i) - \breve{P}(t_i) = T^f(t_i^-) + (T^f)^T(t_i^-) - \breve{P}^f(t_i^-)$$

In general, we expect the correct diagnosis to best mitigate the growth of uncertainty on the system state. Whenever this is not the case, we expect a wrong diagnosis to lead to recurrent detection of the same error. Here we pose $\mathcal{D} = \frac{P(t_i) - \breve{P}(t_i)}{\breve{P}(t_i)}$ and hence look for the hypothesis with minimum trace $tr(\mathcal{D})$.

[1] This is for the UKF, the derivation of the partial minimization linear Kalman gain is given in (Bénazéra & Travé-Massuyès 2007).

[2] The partial filter requires the state projection's partial derivatives, that do not appear in the derivation of the original filter.

1: initialization: $CSD = \{x, I\}$.
2: $(\breve{x}(t_i), \breve{P}(t_i)) \leftarrow$ *Filter(CSD)*.
3: Compute $\delta(\breve{x}(t_i), \breve{P}(t_i))$ and Z^f.
4: **if** there is at least one discrepant observation **then**
5: Compute Λ, Γ and diagnoses (Algorithm 3).
6: Select diagnosis matrix $\Delta^* = \min_\Delta(\mathcal{D}(\Delta))$.
7: If $\Delta^* == 0$ Then *Filter* $\leftarrow$ *UKF*.
8: Else *Filter* $\leftarrow$ *UKF with partial minimization* using Δ^*.

Algorithm 4: Semi-closed loop filter (SCL-UKF).

Fault Detector

We define a simple fault detector based on a Mahalonobis distance which is the statistical distance of a point from a reference mean point. We characterize as discrepant the points that have 99% chances to lie outside $P(t_i^-)$.

Semi-closed loop filter

Our filter closes a loop on sane variables but runs a predictive open loop on unstrusted fragments of the system state. Growing, the uncertainty eventually re-captures the discrepant measures. When this occurs, it is possible to use the additional information to mitigate the growth of the a posteriori error. By scaling the observation space to the re-captured signals, diagnosing, and adapting optimal gains accordingly, we build the SCL-UKF (algorithm 4). This filter uses a mininal state-space isolation in open loop with a scaled a posteriori error minimization in closed loop.

Results

Our case study is the two-link planar robotic arm presented at the beginning of this paper. We used a numerical simulator of the arm movements.

Single fault and hypothesis

First, we study the SCL-UKF on a single fault and hypothesis. Figure 3 pictures its reaction to an incipient change in the second link mass m_2 at step 40 that leads to a discrepant measure of θ_2. The Hitting-Set algorithm produces 21 non-exonerated diagnose. The filter on figure 3 runs on a rejection of the measure of θ_2. Consequently, the filter trusts and closes on the first joint's angular position θ_1 (3(c)). This proves the newly derived gain is able to well decouple the uncertainty since state variables are otherwise tightly coupled. To estimate θ_2, $\dot{\theta}_2$, the SCL-UKF switches between the UKF and the UKF with partial gain (3(a), 3(b)). On the same scenario, a UKF with standard gain closes on the faulty signals with no bulge in the error covariance.

Hypothesis testing

Second, we study the hypothesis testing. Of the 21 diagnose (hypotheses), most correspond to broken influences on the four state variables. Figure 3(d) pictures $tr(\mathcal{D})$ for these four hypotheses and the 35 calls to the UKF with partial gain. Discrimination between $\dot{\theta}_1$ and $\dot{\theta}_2$ is easy: $\dot{\theta}_2$ introduces less uncertainty to the estimate. Hypothesis of a second arm joint positioning failure (θ_2) is eliminated.

Looking at the SCL-UKF as an hypothesis driven self-reconfigurable filter, it wears similarities with Rao-Blackwellized particle filters (RBPF) (Doucet *et al.* 2000) as it selects behavioral hypotheses. However, the RBPF samples hypotheses whereas the SCL-UKF logically draws them from the discrepancies. Also, the RBPF would need around 2^{10} hypotheses and a transition model to capture the arm multiple fault combinations. The SCL-UKF requires partial derivatives for all hypotheses[3], but remains more compact.

Future and related works

We have coupled diagnosis reasoning from first principles with Kalman filtering techniques for nonlinear systems. The result is a novel filter that opens and closes to estimation fragments of its state according to logical selection of diagnosis hypotheses.

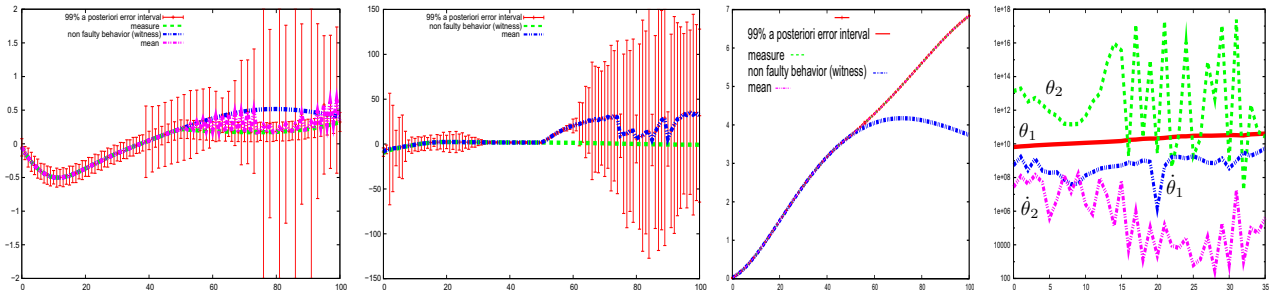
Related works

In (Hofbaur & Williams 2002b) a partial filter is presented that uses a decoupling based on causal and structural analysis of components. However, this scheme only produces independent filters on different subpart of the whole state, as it relies on a bidirectional decoupling of trusted/untrusted state and measured variables. (McIlraith *et al.* 2000) proposes a backward analysis of a causal-graph for producing diagnose and model fitting to adapt to discrepancies. Likewise, adaptive filtering enhances the filter to close on the observations. In that sense, they do not reveal the true uncertainty on the state. We believe that maintaining true uncertainty is key to the efficient control of stochastic systems since it permits for the exploration of a larger but accurately bounded space. While there are no works that we know of about intelligent semi-closed loop Kalman filtering, semi-closed loops have been studied in filtering with numerically bounded uncertainty in (Armengol *et al.* 2000; Benazera, Travé-Massuyès, & Dague 2002). Also, the self-reconfiguration through reasoning from first principles relates to logical filtering (Amir & Russel 2003) as the filtering distributes over disjunctions of the belief state (hypotheses).

Future work and possible extensions

We see at least two extensions to our coupling of diagnosis reasoning and filtering techniques. First, improvements of the RBPF have concentrated on the continuous space and a better use of observations (Hutter & Dearden 2003). However, the RBPF remains limited in the number of modes it can track. We believe that the subset of modes of interest can be reduced by using reasoning and decoupling techniques such as ours, and maintaining a hitting set tree of particle hypotheses for example. Second, we look forward embedding our partial filtering technique into the reinforcement learning framework, for decision and control, and building on existing work (Szita & Lorincz 2004);

[3] It is not too difficult to symbolically or numerically compute the derivatives online.



(a) Angle of the elbow joint (time, θ_2). (b) Angular speed of the elbow joint (time, $\dot{\theta}_2$). (c) Angle of the shoulder joint (time, θ_1). (d) Hypothesis testing (calls, $log(tr(\mathcal{D}))$).

Figure 3: Case study: robotic arm effector mass changes at step 40 while moving its shoulder joint to a reference angle π.

Acknowledgements Emmanuel Benazera is supported by the DFG under contract number SFB/TR-8 (A3).

References

Amir, E., and Russel, S. 2003. Logical filtering. In *IJCAI-03*.

Armengol, J.; Vehi, J.; Travé-Massuyès, L.; and Sainz, M. 2000. Interval model-based fault detection using multiple sliding time windows. In *SAFEPROCESS*, 168–173.

Basseville, M., and Nikiforov, I. V. 1992. *Detection of abrupt changes: theory and application*. Prentice-Hall.

Bénazéra, E., and Travé-Massuyès, L. 2003. The consistency approach to the on-line prediction of hybrid system configurations. In *IFAC Conference on Analysis and Design of Hybrid Systems (ADHS-03)*.

Bénazéra, E., and Travé-Massuyès, L. 2007. A diagnosis driven self-reconfigurable filter (extended). In *Internal Report LAAS-CNRS, Toulouse, France. 9p*.

Benazera, E.; Travé-Massuyès, L.; and Dague, P. 2002. State tracking of uncertain hybrid concurrent systems. In *DX-02*, 106–114.

Biswas, G., and Manders, E.-J. 2006. Integrated system health management to achieve autonomy in complex systems. In *6th Symposium on Fault Detection, Supervision and Safety for Technical Processes*.

Cordier, M.-O.; Dague, P.; Lévy, F.; Montmain, J.; Staroswiecki, M.; and Travé-Massuyès, L. 2004. Conflicts versus analytical redundancy relations : A comparative analysis of the model-based diagnostic approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man and Cybernetics - Part B.* 34(5):2163–2177.

Doucet, A.; de Freitas, N.; Murphy, K.; and Russell, S. 2000. Rao-blackwellised particle filtering for dynamic bayesian networks. In *UAI-00*.

Gautam, G.; Cordier, M.; Lunze, J.; Staroawiecki, M.; and (Eds), L. T.-M. 2004. Diagnosis of complex systems: Bridging the methodologies of the fdi and dx communities. *IEEE Transactions on Systems, Man and Cybernetics - Part B, Special Issue.* 34(5).

Greiner, R.; Smith, B. A.; and Wilkerson, R. W. 1989. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence* 41(1):79–88.

Hamscher, W.; Console, L.; and J. de Kleer, e. 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann.

Hofbaur, M., and Williams, B. 2002a. Mode estimation of probabilistic hybrid systems. *HSCC-2002* 2289:253–266.

Hofbaur, M., and Williams, B. 2002b. Hybrid diagnosis with unknown behavioral modes. In *DX-02*, 97–105.

Hutter, F., and Dearden, R. 2003. The gaussian particle filter for diagnosis of non-linear systems. In *DX-03*.

Julier, S., and Uhlmann, J. 1997. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*.

Kalman, Rudolph, E. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* 82(Series D):35–45.

Levy, F. 1991. Reason maintenance systems and default theories. Technical report, Universit de Paris Nord. Internal report L.I.P.N., http://www-lipn.univ-paris13.fr/ levy/Publications/RMSaDT.pdf, 31p.

McIlraith, S.; Biswas, G.; Clancy, D.; and Gupta, V. 2000. Hybrid systems diagnosis. In *HSCC-2000*.

Narasimhan, S.; Dearden, R.; and Bénazéra, E. 2004. Combining particle filters and consistency-based approaches for monitoring and diagnosis of stochastic hybrid systems. In *DX-04*.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Szita, I., and Lorincz, A. 2004. Kalman filter control embedded into the reinforcement learning framework. *Neural Computation* 16:491–499.

Travé-Massuyès, L., and Calderón-Espinoza, G. 2007. Timed fault diagnosis. In *European Control Conference ECC-07*.

Travé-Massuyès, L.; Escobet, T.; Pons, R.; and Tornil, S. 2001. The Ca-En diagnosis system and its automatic modelling method. *Computacin y Sistemas Journal* 5(2):128–143.

Fault Detection using Interval LPV Models with Uncertain Transport Delay: Application to Canals

Joaquim Blesa, Vicenç Puig, Yolanda Bolea
Automatic Control Department
Universitat Politècnica de Catalunya (UPC)
Pau Gargallo 5, 08028 Barcelona (Spain)
vicenc.puig@upc.edu

Abstract

In this paper, robust fault detection using a passive robust approach based on generating an adaptive threshold considering that the system to be monitored can be described by an interval linear parameter varying (LPV) models with variable transport delay is proposed. This approach allows to consider parameters and associated uncertainty intervals dependence on the operating point. Additionally, a parameter estimation algorithm for interval LPV models with variable transport delay is presented. Finally, a single reach open flow canal, that presents variable transport delay depending on the operating point, has been used to assess the validity of the proposed approach.

Introduction

Distributed parameter systems, considered as systems with a very large number of states, and nonlinear systems could be approximated around a given operating point with low order linear time-invariant (LTI) models in order to use classical fault detection tools (Gertler 1998) (Patton et al. 2000), as usual in control engineering practice. However, simplified LTI parameter models do not preserve any information about the spatial structure and/or non-linearity of the original system and cannot account for it although they can be satisfactory from an input-output point of view (Belforte et al. 2002). Then, some simplified model structures that preserve some information about the non-linearity, the influence of the operating point and the spatial structure of system are needed. Such structure can be provided by linear parameter varying (LPV) models consisting of a linear lumped parameter model in which the parameters and transport delay are not constant but a function of external parameters. In the case system varying-parameters are function of system states and/or operating conditions the model is denoted as a quasi-LPV model or pseudo-LPV model (Rugh and Shamma 2000).

Fault detection methods based on the mathematical model of the plant use the difference between the predicted value from the model and the real value measured by the sensors to detect faults. If this difference is bigger than the threshold, there is a fault in the plant. Otherwise, it is considered that the plant is working properly. However, even in the case of using an LPV model for detecting faults in a canal, there is some modelling error that should be considered in the fault detection procedure. In this paper, the uncertainty will be located in the parameters of the LPV model bounding their values in intervals. Such a model it will called in the following an ***interval LPV model***. The fault detection task will consist as in the case of interval LTI models in checking if the measurement is inside the interval of possible behaviours produced by the interval LPV model (Fagarasan et al. 2004)(Puig et al. 2002)(Sainz et al. 2002).

The paper is organized as follows. In Section "Interval LPV Model Fault Detection", a method for using interval LPV models in fault detection is introduced. Section "Interval LPV Model Identification" presents a method for the identification of interval LPV models. In Section "Application to a Test Bench Canal", modelling, identification and fault detection based on the interval LPV models methodologies to an open canal are presented to show its effectiveness. Finally, in Section "Conclusions" the main conclusions are presented.

Interval LPV Model Fault Detection

Interval LPV Model

Let us consider that the process to be monitored can be represented as a linear parameter varying system in discrete-time by the following input-output relationship without considering faults, disturbances and noise:

$$y(k) = M(q,\theta(p_k))u(k) = \frac{B(q,\theta(p_k))}{A(q,\theta(p_k))}u(k) \quad (1)$$

where: $u(k)$ is the input, $y(k)$ is the output, $M(q,\theta(p_k))$ is the transfer function with numerator $B(q,\theta(p_k))$ and denominator $A(q,\theta(p_k))$ in terms of the classical q-operator. Since an LPV model is essentially a parameterized family of LTI models, the parameters $\theta(p_k)$ vary according to the process operating point described by such measurable process variable p_k following some known function:

$$\theta(p_k) = f(p_k) \tag{2}$$

Then, the numerator and denominator polynomials of (1) can be written as a function of p_k as

$$B(q,p_k) = b_0(p_k) + b_1(p_k)q^{-1} + \ldots + b_{n_b}(p_k)q^{-n_b}$$
$$A(q,p_k) = 1 + a_1(p_k)q^{-1} + \ldots + a_{n_a}(p_k)q^{-n_a} \tag{3}$$

However, uncertainties in the parameters are always present in the model of the actual system. Taking parametric uncertainties into account, the healthy system model should include a vector of uncertain parameters $\theta(p_k) \in [\theta(p_k)]$, where $[\theta(p_k)]$ is a bounded set of box (interval) type such that each component $\theta(p_k)_i \in [\underline{\theta}(p_k)_i, \overline{\theta}(p_k)_i]$ $i=1,..n_p$, being n_p the number of uncertain parameters. This set contains all possible values of $\theta(p_k)$ when the system operates normally. The resulting model is known as an ***interval model***.

Interval LPV Fault Detection

The principle of model-based fault detection is to test whether the measured input and output from the system lie within the behaviour described by a model of the faultless system. If the measurements are inconsistent with the model of the faultless system, the existence of a fault is proved. The residual usually describes the consistency check between the predicted, $\hat{y}(k)$, and the real behaviour, $y(k)$, as:

$$r(k) = y(k) - \hat{y}(k) \tag{4}$$

According to (Gertler 1998), given a system described by Eq. (1), a general form of the predicted behaviour is

$$\hat{y}(k) = G_u(q,\theta(p_k))u(k) + G_y(q,\theta(p_k))y(k) \tag{5}$$

that includes as special cases:

- ***simulation*** (i.e. no correction of the modelled behaviour with sensor measurements is introduced):
  $G_u(q,\theta(p_k)) = M(q,\theta(p_k))$, $G_y(q,\theta(p_k)) = 0$
- ***prediction*** (i.e. complete correction of the modelled behaviour with sensor measurements):
  $G_u(q,\theta(p_k)) = B(q,\theta(p_k))$,
  $G_y(q,\theta(p_k)) = 1 - A(q,\theta(p_k))$
- or ***observation*** (i.e., partial correction of the modelled behaviour with sensor measurements through the observer gain $K(q^{-1}) = \sum_{i=1}^{n_a} k_i q^{-i}$)

$$G_u(q,\theta(p_k)) = \frac{B(q,\theta(p_k))}{1 - A(q,\theta(p_k)) + K(q,\theta(p_k))},$$
$$G_y(q,\theta(p_k)) = \frac{K(q,\theta(p_k))}{1 - A(q,\theta(p_k)) + K(q,\theta(p_k))}$$

When using such general formulation for predicted behavior, the residual generation structure for Eq. (4) can be graphically represented as in Fig. 1.

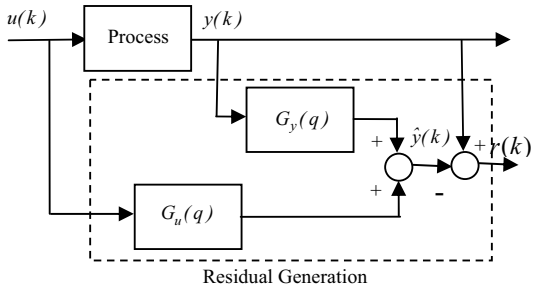


Fig. 1. General residual generation structure.

Ideally, residuals should only be affected by the faults. However, the presence of disturbances, noise and modelling errors causes residuals to become nonzero and thus interferes with the detection of faults. Therefore, the fault detection procedure must be ***robust*** against these undesired effects (Chen and Patton 1999). In case of modelling a dynamic system using an interval model, the predicted output behaviour is described by a set that can be bounded at any iteration by an interval $[\hat{y}(k)]$. whose bounds $[\underline{\hat{y}}(k), \overline{\hat{y}}(k)]$ are computed by solving the two optimization problems:

$$\underline{\hat{y}}(k) = \min_{\theta \in [\underline{\theta}, \overline{\theta}]} \left(G_u(q,\theta)u(k) + G_y(q,\theta)y(k) \right)$$
$$\overline{\hat{y}}(k) = \max_{\theta \in [\underline{\theta}, \overline{\theta}]} \left(G_u(q,\theta)u(k) + G_y(q,\theta)y(k) \right) \tag{6}$$

Then, fault detection test is based on propagating the parameter uncertainty to the residual and checking if:

$$0 \in [r(k)] = y(k) - [\hat{y}(k)] \quad \text{or, } y(k) \in [\hat{y}(k)] \tag{7}$$

holds or not. In case it does not hold a fault can be indicated.

Interval LPV Prediction

In this paper, only the prediction case is considered. Using Eq. (5) and (3), predicted output $\hat{y}(k)$ can be calculated by

$$\hat{y}(k) = \sum_{i=0}^{n_b} b_i(p_k)u(k-i) - \sum_{j=1}^{n_a} a_j(p_k)y(k-j) \tag{8}$$

in function of the LPV parameters (2), input data *u(k-i)* for $i=0,\ldots, n_b$ and output measures *y(k-j)*, $j=1,\ldots,n_a$.

The interval for the predicted output (6) can be evaluated using (8) by considering parametric uncertainty

$$\bar{\hat{y}}(k) = \max\left(\sum_{i=0}^{nb} b_i(p_k)u(k-i) - \sum_{j=1}^{na} a_j(p_k)y(k-j)\right)$$
$$\underline{\hat{y}}(k) = \min\left(\sum_{i=0}^{nb} b_i(p_k)u(k-i) - \sum_{j=1}^{na} a_j(p_k)y(k-j)\right) \quad (9)$$

where $b_i(p_k) \in [b_i(p_k)]$ and $a_j(p_k) \in [a_j(p_k)]$

In order to solve optimisation problems in (9), predicted output (8) can alternatively be written in regressor form as

$$\hat{y}(k) = \phi(k)\left[\theta(p_k)\right] \quad (10)$$

where

$$\phi(k) = \left[-y(k-1),....-y(k-na),u(k),...,u(k-nb)\right]$$

$$[\theta(p_k)] = \left[[a_1(p_k)],\cdots,[a_{na}(p_k)],[b_0(p_k)],\cdots,[b_{nb}(p_k)]\right]^T$$

with:

$$[a_j(p_k)] = [\underline{a_j}(p_k), \overline{a_j}(p_k)] \text{ with } j=1,...,n_a$$
$$[b_i(p_k)] = [\underline{b_i}(p_k), \overline{b_i}(p_k)] \text{ with } i=0,...,n_b$$

Let us consider that uncertain parameters can be parametrised considering nominal parameters plus associated uncertainty as follows:

$$[a_j(p_k)] = [a_j^0(p_k) - \lambda_j, a_j^0(p_k) + \lambda_j], \quad j=1,...,n_a$$
$$[b_i(p_k)] = [b_i^0(p_k) - \lambda_{i+n_a+1}, b_i^0(p_k) + \lambda_{i+n_a+1}], \quad i=0,...,n_b$$

where $\forall j\ \lambda_j \in \Re^+$ and $\forall i\ \lambda_{i+n_a+1} \in \Re^+$

or, in vector/matrix form as

$$[\theta(p_k)] = \theta^o(p_k) + T[v] \quad (11)$$

where $\theta^0(p_k) \in \Re^{n_p}$ with $n_p = n_a + n_b + 1$ is defined as in (5), T is defined as a $n_p \times n_p$ diagonal matrix

$$T = \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_{np} \end{pmatrix} \quad (12)$$

and $[v]$ is a $n_p \times 1$ interval vector

$$[v] = ([v_1],[v_2],...,[v_{np}])^T \quad (13)$$

where $\forall k \in \{1,...,n_p\}\ [v_k] = [-1,1]$.

Then, using the results of interval arithmetic (Mo and Norton 1988 Neumanier 1990) allows to evaluate (9) as follows

$$\bar{\hat{y}}(k) = \phi(k)\theta^o(p_k) + \left\|\phi(k)T\right\|_1$$
$$\underline{\hat{y}}(k) = \phi(k)\theta^o(p_k) - \left\|\phi(k)T\right\|_1 \quad (14)$$

Uncertainty Transport Delay in Fault Detection

Introducing now pure transport delay in model (8), the output prediction for a given delay $\tau(p_k)$ can be expressed as

$$\hat{y}(k,\lfloor\tau(p_k)\rfloor) = \sum_{i=0}^{n_b} b_i(p_k)u(k-i-\lfloor\tau(p_k)\rfloor) - \sum_{j=1}^{n_a} a_j(p_k)y(k-j) \quad (15)$$

and alternative in vector/matrix form (10)

$$\hat{y}(k) = \phi(k,\lfloor\tau(p_k)\rfloor)\left[\theta(p_k)\right] \quad (16)$$

where $\lfloor\ \rfloor$ denotes the nearest integer towards minus infinity.

Considering uncertainty in delay:

$$\tau(p_k) \in [\tau_0(p_k) - \lambda_\tau, \tau_0(p_k) + \lambda_\tau]$$
where $\tau_0(p_k) > \lambda_\tau$ and $\tau_0(p_k), \lambda_\tau \in \Re^+$

then, the interval for the predicted output given by (15) can be expressed

$$\bar{\hat{y}}(k) = \max\left(\bar{\hat{y}}(k,\lfloor\tau(p_k)-\lambda_\tau\rfloor),...,\bar{\hat{y}}(k,\lfloor\tau(p_k)+\lambda_\tau\rfloor)\right)$$
$$\underline{\hat{y}}(k) = \min\left(\underline{\hat{y}}(k,\lfloor\tau(p_k)-\lambda_\tau\rfloor),...,\underline{\hat{y}}(k,\lfloor\tau(p_k)+\lambda_\tau\rfloor)\right) \quad (17)$$

Remark: notice that in the previous optimization problems $\bar{\hat{y}}(k,\tau(p_k))$ and $\underline{\hat{y}}(k,\tau(p_k))$ have to be evaluated as in equation (9) using (15) for n_τ possible delays, where

$$n_\tau = \lfloor\tau(p_k)+\lambda_\tau)\rfloor - \lfloor\tau(p_k)-\lambda_\tau)\rfloor$$

Interval LPV Model Identification

LPV Model Identification

One of the key points in model based fault detection is how models are built and its uncertainty is estimated. Since the LPV model is essentially a parameterized family of LTI models, a possible identification scheme is to fix each operating point, and collect enough data to identify the LTI model at that point. The identified LTI coefficients can then be used as interpolation points to find the coefficients as polynomial functions of the parameter p_k. Alternatively, using LPV identification methods as those proposed by Bamieh (Bamieh and Giarré 2002), this process can be carried out in 'one shot'. This first method would produce a similar model that the second but it would require at least *n*-LTI models at distinct set points. For this reason in order to estimate the parameters of model (1), the second method is used. Such a method allows estimating discrete-time LPV model parameterized as follows (Puig et al. 2005): $n_p = n_a + n_b + 1$ is the number of parametric functions (2) to be identified that are considered to be polynomials in p_k of order *N*-1, i.e.,

$$a_j(p_k) = a_{j1}^0 + a_{j2}^0 p_k + ... + a_{jN}^0 {p_k}^{N-1}$$
$$b_i(p_k) = b_{i1}^0 + b_{i2}^0 p_k + ... + b_{iN}^0 {p_k}^{N-1}. \quad (18)$$

The parameter estimation problem can be easily formulated in linear regression form. According to Bamieh (Bamieh and Giarré 2002), least squares algorithm can be applied to this problem by introducing an extended regressor ψ_k defined by

$$\Psi_k := \phi_k \pi_k = \begin{bmatrix} -y(k-1) \\ \vdots \\ -y(k-n_a) \\ u(k) \\ \vdots \\ u(k-n_b) \end{bmatrix} \begin{bmatrix} 1 & p_k & \cdots & p_k^{N-1} \end{bmatrix} \quad (19)$$

then,

$$y(k) = \langle \Theta, \Psi_k \rangle \quad (20)$$

where: $\langle A, B \rangle := trace(A^* B)$ is the inner product of the matrices A and B of equal dimensions, A^* is the complex conjugate transpose of the matrix A and the matrix Θ contains all the coefficients to be identified

$$\Theta := \begin{bmatrix} a_{11}^0 & \cdots & a_{1N}^0 \\ a_{21}^0 & \cdots & a_{2N}^0 \\ \vdots & \vdots & \vdots \\ a_{n_a 1}^0 & \cdots & a_{n_a N}^0 \\ b_{01}^0 & \cdots & b_{0N}^0 \\ \vdots & \vdots & \vdots \\ b_{n_b 1}^0 & \cdots & b_{n_b N}^0 \end{bmatrix} \quad (21)$$

Algorithm for Interval LPV Model Identification

Let us consider M measurements of $y(k)$ and $\phi(k)$ from a scenario free of faults and rich enough from the identifiability point of view. The aim is to estimate interval for uncertain LPV parameters of model described by a vector $[\theta(p_k)]$ parametrised as in (11), such that

$$\forall k \in \{1,...,M\} \quad y(k) \in \left[\underline{\hat{y}}(k), \overline{\hat{y}}(k) \right] \quad (22)$$

where $\underline{\hat{y}}(k)$ and $\overline{\hat{y}}(k)$ are computed using (14).

LPV interval model estimation is a two-phase process. First nominal LPV parameters $\theta^o(p_k)$ are estimated using previous parameter estimation algorithm. Second, the width of the interval around this nominal parameters given by

$$T = \lambda \begin{pmatrix} \lambda_{01} & 0 & ... & 0 \\ 0 & \lambda_{02} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_{0np} \end{pmatrix} = \lambda T_0 \quad (23)$$

where $\lambda_{01}, \lambda_{02}, ..., \lambda_{0np}$ are constants determined with nominal parameters of LPV model and adjusted using an optimization process as proposed by Ploix in the case of LTI interval models (Ploix et al. 1999). The problem of computing intervals for uncertain parameters can be reduced to calculate λ.

Using (23) in (14) allows to rewrite Eq. (22) alternatively as

$$-\lambda \left\| \phi(k) T_0 \right\|_1 \le y(k) - \phi(k)\theta^o(p_k) \le \lambda \left\| \phi(k) T_0 \right\|_1 \quad (24)$$

$\forall k \in \{1,..., M\}$, what allows to derive

$$\lambda \ge \max \left(\frac{y(k) - \phi(k)\theta^o(p_k)}{\left\| \phi(k) T_0 \right\|_1}, \frac{\phi(k)\theta^o(p_k) - y(k)}{\left\| \phi(k) T_0 \right\|_1} \right) \quad (25)$$

In order to select a particular value of λ, an optimisation criteria that measure how fit the interval model cover the data is proposed. Since at each time instant k, the width of interval prediction given by Eq. (14) is $2\left\| \phi(k) T(\lambda) \right\|_1$, λ will be selected such that the following criteria:

$$J = 2 \sum_{k=1}^{M} \left\| \phi(k) T(\lambda) \right\|_1 \quad (26)$$

is minimised. This leads to take

$$\lambda = \sup_{k \in [1,...,M]} \left(\max \left(\frac{y(k) - \phi(k)\theta^0(p_k)}{\left\| \phi(k) T_0 \right\|_1}, \frac{\phi(k)\theta^0(p_k) - y(k)}{\left\| \phi(k) T_0 \right\|_1} \right) \right) \quad (27)$$

allowing to obtain the intervals for uncertain LPV parameters.

Interval Transport Delay in Model Identification

Nominal $\tau_0(p_k)$ delay and its uncertainty λ_τ can be calculated by interpolation time delays estimated at different operation points by the correlation method that is based in the impulse response by statistical method. Considering that the input process signal is a white noise and carrying out the study of the independence between the input and output process signals using confidence intervals (usually, 99% or 95%) an interval for delay is estimated (Isermann and Bauer 1974)(Söderström and Stoica 1989).

Then extended regressor ψ_k (19) used to calculate nominal LPV model with nominal delay $\tau_0(p_k)$ results

$$\Psi_k := \phi_k \pi_k = \begin{bmatrix} -y(k-1) \\ \vdots \\ -y(k-n_a) \\ u(k-\tau_0(p_k)) \\ \vdots \\ u(k-n_b-\tau_0(p_k)) \end{bmatrix} \begin{bmatrix} 1 & p_k & \cdots & p_k^{N-1} \end{bmatrix} \tag{28}$$

And the optimisation problem utilized to calculate uncertainty λ of parameters $a_j(p_k)$ and $b_i(p_k)$

$$\lambda = \sup_{k\in[1,\ldots,M]} \left(\max \left(\begin{array}{l} \inf \left(\dfrac{y(k)-\phi(k,\lfloor \tau(p_k)-\lambda_\tau \rfloor)\theta^0(p_k)}{\left\|\phi(k,\lfloor \tau(p_k)-\lambda_\tau \rfloor)T_0\right\|_1}, \ldots \right. \\ \left. \dfrac{y(k)-\phi(k,\lfloor \tau(p_k)+\lambda_\tau \rfloor)\theta^0(p_k)}{\left\|\phi(k,\lfloor \tau(p_k)+\lambda_\tau \rfloor)T_0\right\|_1} \right) \\ \inf \left(\dfrac{\phi(k,\lfloor \tau(p_k)-\lambda_\tau \rfloor)\theta^0(p_k)-y(k)}{\left\|\phi(k,\lfloor \tau(p_k)-\lambda_\tau \rfloor)T_0\right\|_1}, \ldots \right. \\ \left. \dfrac{\phi(k,\lfloor \tau(p_k)+\lambda_\tau \rfloor)\theta^0(p_k)-y(k)}{\left\|\phi(k,\lfloor \tau(p_k)+\lambda_\tau \rfloor)T_0\right\|_1} \right) \end{array} \right) \tag{29}$$

Remark: Like interval prediction considering uncertain time delay, algorithm (29) has to evaluate the n_τ different models with possible delays and choose the best model at every time *k*.

Application to a Test Bench Canal

Motivation

Water is a precious resource that should be managed efficiently. Irrigation is the main water consuming activity around the world, as it represents about 80% of the available fresh water consumption. It is widely accepted that automation could lead to a better efficiency of water management in many irrigation systems with open canals, which are subject to large losses. However, open canal systems are not easy to control, as they are large distributed systems with complex dynamics. Besides they involve mass energy transport phenomena which behave as intrinsically distributed parameter systems. Their complete dynamics is represented by non-linear partial differential hyperbolic equations (PDE) that are function of time as well as of spatial coordinates: Saint-Venant's equations. This equation system has no known analytical solution in real geometry and it has to be solved numerically (characteristic method, Preissmann implicit scheme, etc.) (Cunge et al. 1980). The resulting simulation models are therefore suitable for scientific and time-consuming simulations but are too complex for on-line fault-detection. Alternatively, a simplified model for fault detection relates the upstream and downstream level will be proposed in next section.

Canal Description

A single pool equipped with an upstream sluice gate and a downstream spillway composes the test canal used in this study. An electromotor is driving gate position and two sensors located upstream and downstream of the gate are measuring the flows. Upstream of this gate there is a damming of constant level H=3.5m. The total length of the pool is L=2km, with an initial flow Q_0=1m^3/s, a gate discharge coefficient C_{dg}= 0.6, a Manning roughness coefficient n=0.014, gate width and canal width B=2.5m, a downstream spillway of height y_s = 0.7m, a spillway coefficient C_{ds}= 2.66, and a bottom slope I_0= 5.10^{-4}.

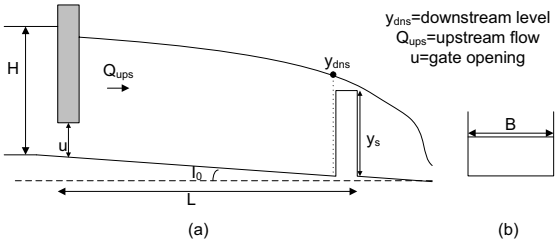


Fig.2. Canal scheme.(a) Longitudinal and (b) cross section.

According to Bolea (Bolea et al. 2004), an LPV model with a first order differential equation with time delay is able to describe canal dynamics at different operating points depending on gate position

$$y_{dns}(t) + T(u)\frac{dy_{dns}(t)}{dt} = K(u)u(t-\tau(u)) \tag{30}$$

where y_{dns} and u are the downstream level and the position of the upstream gate. The LPV parameters of the physical LPV model (which it is represented by a FOPDT model) are: the theoretical estimated steady-state gain $K(u)$, the estimated time constant $T(u)$, and the estimated delay $\tau(u)$.

The results presented in this paper are based on a simulator developed by the group of "*Modelling and Control of Hydraulic Systems*" at the UPC (Bolea and Blesa, 2000) that solves numerically Saint-Venant's equations which describe the water dynamics accurately.

LPV Canal Model

Discretising the model given by (30) using the ZOH transform method, the following discrete LPV model $G_{LPV}(z)$ is obtained (Åström and Wittenmark 1989)

$$G_{LPV}(z) = \frac{b_0(u)z}{z - a_1(u)} z^{-\tau_0(u)} \tag{31}$$

using the sample time T_s, we have

$$a(u)=e^{\frac{-T_s}{T(u)}}, b(u)=K(u)(1-a(u)), \tau_0(u)=\left\lfloor \frac{\tau(u)}{T_s} \right\rfloor = n \quad (32)$$

The discrete-time model (31) can be expressed in time prediction form

$$\begin{aligned}
&\hat{y}(k)=a_1(k-1)y(k-1)+b_0(k-1)u(k-\tau_0(k-1)) \\
&a_1(k)=a_{10}+a_{11}p_k+a_{12}p_k^2 \\
&b_0(k)=b_{00}+b_{01}p_k+b_{02}p_k^2 \\
&\tau_0(k)=\tau_0+\tau_1 p_k+\tau_2 p_k^2 \qquad (33)
\end{aligned}$$

where the variation of the two model parameters a_1, b_0 and the transport delay τ_0 with the operating point p_k have been approximated by a second order polynomial. Here the operating point is considered to be characterized by the input (gate opening), i.e., $p_k=u(k)$. In next section, parameters of polynomials in Eq. (33) will be estimated using input/data registered from the canal.

Model Identification

The input signal applied to the canal used to estimate and validate the LPV model (33) is a set of steps that sweeps all the operating points (gate opening from 0.1 to 0.9 meters) (Fig. 3).

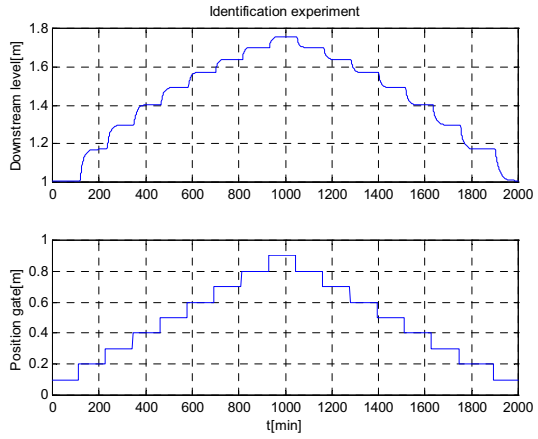


Fig. 3. Input and output signal for identification

The nominal delay of the system τ and its associated uncertainty is obtained at each operating point by the method described in Section "Interval Transport Delay in Model Identification". The variation of the transport delay with the operating can be approximated by a second order polynomial (coefficients τ_i) (Table 1 and Fig. 4) . The uncertainty of the time delay obtained is λ_τ=0.71 s.

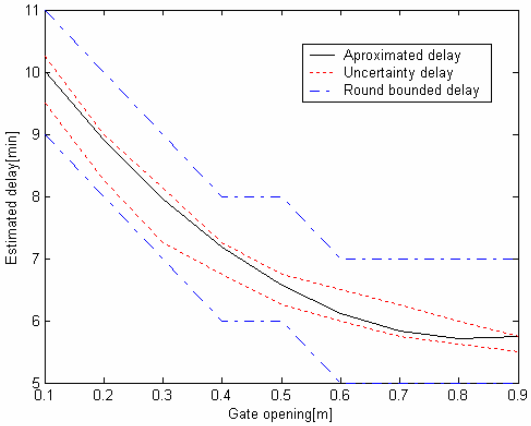


Fig. 4. Approximated delay between interval delay estimated by correlation and round bounded delay.

Applying the LPV least squares method, described in Section "LPV Model Identification", to the input and output data and using the estimated LPV time delay, coefficients a_1 and b_0 can be determined (Table 1).

	1	$p(k)$	$p^2(k)$
a_1^0	-0.991	0.341	-0.197
b_0^0	0.397	0.081	-0.206
τ_0	11.3	-13.6	8.2

Table 1. Coefficients of the estimated LPV model.

In order to validate the LPV model, data registered from the canals and estimated output are compared. As it can be seen in Fig. 5 the accuracy of LPV model is very good as in prediction as in simulation.

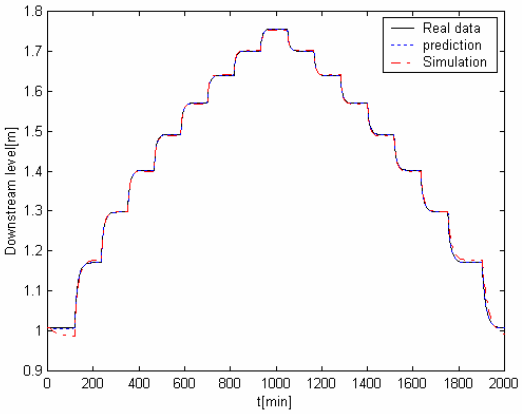


Fig. 5. Real data, prediction and simulation (LPV model).

Using the nominal LPV (33) with parameters Table 1, and the set of input/output data, intervals bounding uncertain parameters can be calculated applying the method presented in Section "Algorithm for Interval LPV Model Identification".

In order to justify the use and benefits of using an LPV model for fault detection in this case, instead of an LTI model, estimated intervals for the LPV model are compared with those of an LTI model obtained in the mean operating point applying method presented in Section "Algorithm for Interval LPV Model Identification" (Table 2). Uncertainty of LPV parameters is smaller than those of LTI parameters. This is due to LPV model represents better the canal (non-linear system) than LTI model, as discussed before.

	$\lambda_1(a_1)$	$\lambda_2(b_1)$	λ_τ
LTI model	0.0798	0.0295	-
LPV model	0.0096	0.0034	0.71

Table 2. Confidence for parameters LTI and LPV intervalar models.

In Fig. 6 and 7 parameters a and b and its associated uncertainty are presented in case of LPV and LTI models.

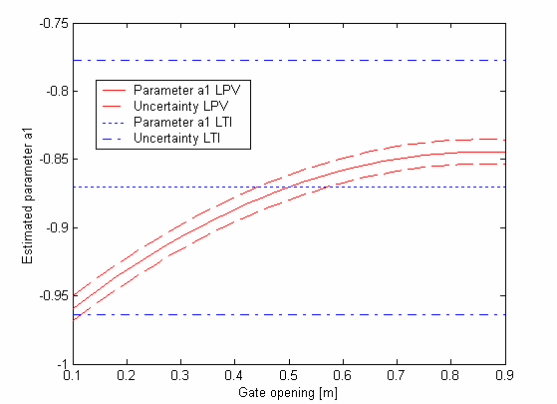


Fig. 6. LPV and LTI parameter a_1 with its intervals.

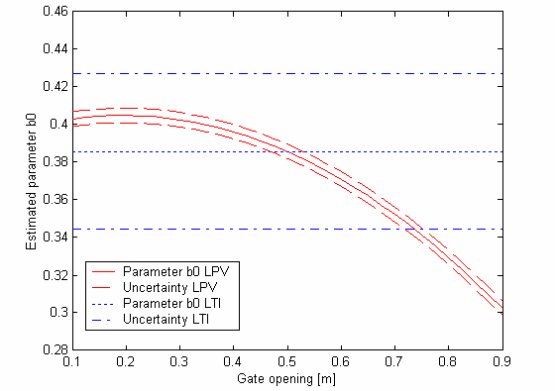


Fig. 7. LPV and LTI parameter b_0 with the intervals of uncertainty

LPV Fault Detection

Once LPV interval models have been found, interval prediction can be computed as it was described in Section "Uncertainty Transport Delay in Fault Fetection" and applied to fault detection.

Fig 8 shows interval prediction for an LPV interval model and for an LTI model in a fault free scenario. Interval prediction corresponding to LPV model fits better measured output. This will allow the LPV model to detect faults than LTI model would not be able.

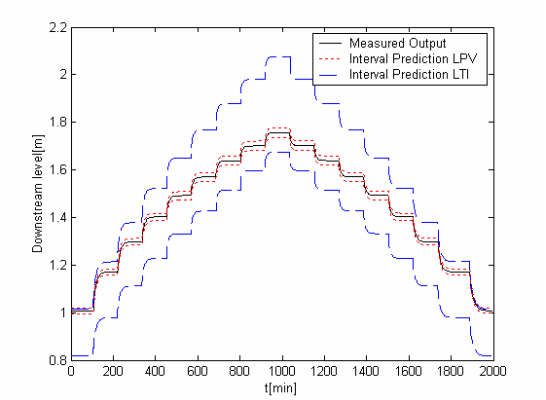


Fig. 8. Measured output and interval predictions (LPV and LTI) in a 'Fault Free' scenario.

In order to compare the behaviour of the two models in fault detection, two kinds of faults has been simulated: offset (additive) faults in input and output sensors.

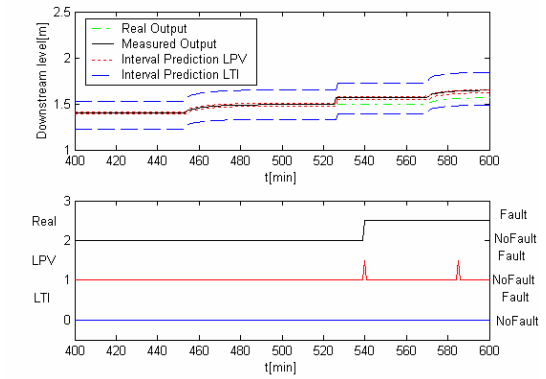


Fig. 9. Measured output, real output, interval prediction and fault indication (LPV and LTI) in an offset output sensor fault scenario at t=540min.

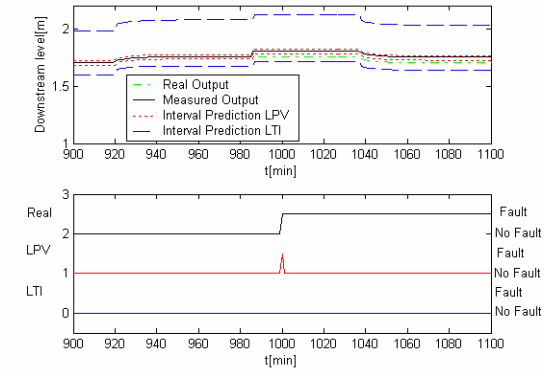


Fig. 10.Measured output, real output, interval prediction and fault indication (LPV and LTI) in an offset output sensor fault scenario at t=1000min.

Fig 9 and Fig 10 show the behaviour of the two models in an offset output sensor fault scenario. In Fig 9, an offset fault of 8 cm in sensor y occurs at time 540 min. In Fig 10, an offset fault of 5 cm in sensor y occurs at time 1000 min. In the two cases, using the LPV model faults are

detected while using LTI model are not. It should be noticed that the persistency of fault is only one step. This is due to both models works using one step-ahead prediction.

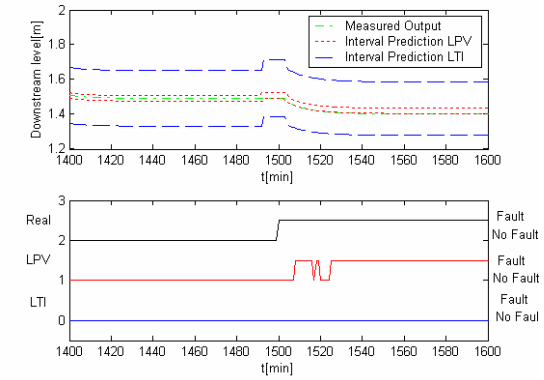


Fig. 11. Measured output, interval prediction and fault indication (LPV and LTI) in an offset input sensor fault scenario at t=1500min.

Fig 11 shows the behaviour of the two models (LTI and LPV) in an offset input sensor fault scenario. A fault of 15 cm in sensor *u* occurs at time 1500. LPV model is able to detect the fault while the LTI is not.

In Table 3, the behaviour of the two models in fault detection is summarized.

	LTI model	LPV model
Offset Fault y	>15cm	>2cm
Offset Fault u	>25cm	>5cm

Table 3. Fault Detection behavior of LTI and LPV models

Conclusions

In this paper, the identification and fault detection using linear parameter varying (LPV) interval models has been presented and applied to an open-flow canal. The use of such models has been motivated because the canal parameters and transport delay depend on the operating point. The fault detection is based on a passive robust approach based on interval methods. Finally, the proposed interval LPV identification and fault detection algorithms have been applied to a test-bench canal obtaining better results than using an LTI interval model.

Acknowledgments

This work has been funded by the grant CICYT DPI-2005-04722 of Spanish Ministry of Education and by Research Comission of the Generalitat de Catalunya (group SAC ref. 2005SGR00537).

References

Åström, K.J., and Wittenmark B. eds. 1989. *Adaptive Control*. Prentice-Hall.

Bamieh, B., and Giarré, L. 2002. Identification of linear parameter varying models. *Int. J. Robust Nonlinear Control* vol.12, pp.841-853.

Belforte, G., Dabbene, F., and Gay, P. 2002. LPV Approximation of Distributed Parameter Systems in Environmental Modeling. *International Environmental Modelling and Software Society Proc*. vol 2.

Bolea, Y., and Blesa, J. 2000. Irrigation canal simulator (ICS). Automatic Control Dept., Technical Univ. of Catalonia. Internal Report.

Bolea, Y., Puig, V., Blesa, J., Gómez M., and Rodellar, J. 2004. An LPV Model for Canal Control. In Proccedings 10th IEEE Int. Conf. on Methods and Models in Automation and Robotics MMAR.

Chen, J., and Patton, R.J. eds 1999. *Robust Model-Based Fault Diagnosis for Dynamic System*. Kluwer Academic Publishers.

Cunge, J.A., Holly, F.M., and Holly, A. eds. 1980. *Practical aspects of computational river hydraulics*, Pitman Advanced Publishing Program.

Fagarasan, I., Ploix, S., and Gentil, S. 2004. Causal fault detection and isolation based on a set-membership approach. *Automatica*. pp. 2099-2110.

Gertler, J.J. eds. 1998. *Fault Detection and Diagnosis in Engineering Systems.* Marcel Dekker.

Isermann, R., and Bauer, U. 1974. Two-Step Process Identification with Correlation Analysis and Least-Squares Parameter Estimation. *Journal of Dynamic Systems, Measurement and Control*. dec. 1974, pp. 426-432.

Mo S.H., and Norton, J.P. 1988. Recursive parameter-bounding algorithms which compute polytope bounds. *In procceding 2nd MACS 121h World Congress on scientific computation*, Paris. pp.127-132.

Neumaier, A. eds. 1990. *Interval methods for systems of equations*. Cambridge University Press.

Patton, R.J., Frank, P.M., and Clark, R.N. eds 2000. *Issues of Fault Diagnosis for Dynamic Systems*. Springer.

Ploix, S., Adrot, O., and Ragot, J. 1999. Parameter Uncertainty Computation in Static Linear Models. *In proccedings 38th IEEE Conference on Decision and Control*, Phoenix, Arizona, USA.

Puig, V., Quevedo, J., Escobet, T., and De las Heras, S. 2002. Robust Fault Detection Approaches using Interval Models. *In proccedings IFAC World Congress (b'02)*, Barcelona, Spain.

Puig, V., Quevedo, J., Escobet, T., Charbonnaud, P., and Duviella., E. 2005. Identification and Control of an Open-flow Canal using LPV Models. *In proccedings 44th IEEE Conference on Decision and Control and European Control Conference*, Sevilla, Spain.

Rugh W., and Shamma, J. 2000. Research on gain scheduling. *Automatica*, vol.36, pp.1401-1425.

Sainz, M. Á., Armengol, J., and Vehí, J. 2002. Fault detection and isolation of the three-tank system using the modal interval analysis. *Journal of Process Control*, Vol.12, Issue 2. Pages 325-338.

Söderström, T., and Stoica, P. eds 1989. *System Identification*. Prentice-Hall.

Distributed Chronicles for On-line Diagnosis of Web Services

Marie-Odile Cordier and **Xavier Le Guillou** and **Sophie Robin** and **Laurence Rozé** and **Thierry Vidal**
IRISA - Institut de Recherche en Informatique et Systèmes Aléatoires
INSA - INRIA - Université de Rennes 1
Campus de Beaulieu
35042 Rennes Cédex – FRANCE

Abstract

The formalism of chronicles has been proposed a few years ago to monitor and diagnose dynamic physical systems. A chronicle, expressed by a set of time-constrained events, describes the situations to monitor. Efficient algorithms have been proposed to analyze a flow of observed events and recognize the corresponding chronicles on line.

It is now well known that, due to the important size of a global model, distributed approaches are better suited to monitor actual systems. In this article, we show how to adapt the chronicle-based approach to a distributed context. We propose a decentralized architecture and describe a high-level algorithm which is in charge of synchronizing the local diagnoses, computed by chronicle-based local diagnosers, and merging them into a global diagnosis. We illustrate this article with a simple example.

This work is motivated by an application that aims at monitoring the behavior of software components, especially web services. In this context, a request is sent to a web service which collaborates with other services to provide the adequate reply. Faults may propagate from one service to another and diagnosing them is a crucial issue, in order to react properly. This work takes place within the context of the WS-Diamond European project.

1 Introduction

Monitoring and diagnosing dynamic systems have become a very active topic in research and development for a few years. Applications we are aiming at, usually deal with monitoring industrial systems (Cordier *et al.* 1998; Pencolé *et al.* 2002; Aguilar *et al.* 1994), or doing medical monitoring (Quiniou *et al.* 2001; Dojat *et al.* 1998). Contrary to expert approaches that rely on the knowledge acquired by a diagnosis practitioner, model-based approaches rely on a behavior model of the monitored system. The main formalisms used to describe dynamic systems are continuous models based on differential equations, essentially used in control theory, and discrete event systems based on finite state machines (automata, Petri nets, ...). A formalism commonly used for on-line monitoring, in particular by people from the artificial intelligence community, is the one of chronicles. This formalism, proposed by Dousson (Dousson *et al.* 1993) has been, since it was created, widely used and extended (Cordier *et al.* 1998; Dojat *et al.* 1998; Cordier & Dousson 2000). A chronicle describes a situation that is worth identifying within the diagnosis context. It is made up a set of events and temporal constraints between those events. As a consequence, this formalism fits particularly well problems that consider a temporal dimension. The set of interesting chronicles constitutes the base of chronicles. Then, monitoring the system consists of analyzing flows of events, and recognizing on fly patterns described by the base of chronicles.

One of the key issues of model-based approaches is that the model is generally too complex to be globally described. Distributed or decentralized approaches were proposed (Baroni *et al.* 2000; Debouk *et al.* 2000; Aghasaryan *et al.* 1998; Jiroveanu & Boël 2005; Pencolé & Cordier 2005), which main idea was to consider the system as a set of independent components instead of a unique entity. Then, the behavior of the system was described by a set of local models of each component and by the synchronization constraints between the components. Considering chronicle-based approaches, to our knowldegde, few distributed approaches exist. It is this, that motivated the work we present in this article. The contribution of this paper consists of adapting the chronicle-based approach to distributed systems.

We first describe (section 2) the architecture of a diagnosis and monitoring system that uses the proposed formalism. In section 3, we detail the example that will be used all along this paper. Then, we present (section 4) an extension of the classical formalism of chronicles that allows expressing synchronization constraints between the different components of the system. A simplified algorithm dedicated to the computation of the global diagnosis is proposed in section 5. It explains how local diagnoses computed by each component may be synchronized to compute a global diagnosis. We finish (section 6) by comparing our work with related work and presenting perspectives.

The application that induced this work is the monitoring of software components, and more precisely of web services (Ardissono *et al.* 2005; Barbon *et al.* 2006; Baresi *et al.* 2004; Salaün *et al.* 2004; Lazovik *et al.* 2003), within the context of the WS-DIAMOND (Web Service DIAgnosability, MONitoring and Diagnosis) European project. It consists of checking that a request sent to a web

service is correctly processed by the set of involved services, detecting a potential failure and elaborating a diagnosis in order to react accordingly, by compensation of reconfiguration actions. The chronicles we are using describe the normal and abnormal behaviors of each web service involved in the process, and the communications with other services. It is common that a fault occurring on a service propagates, via communication links, to other services, causing its primary effects later, on a service indirectly responsible for the problem. This is an interesting problem towards diagnosis.

2 General architecture of the diagnosis system

We chose to use a decentralized architecture. Indeed, in our approach, diagnoses are computed locally but the synchronization of those local diagnoses and the computation of the global one are centralized, which favors coordinated decision-making. This architecture contrasts with a distributed architecture where the global diagnosis would be elaborated directly by communicating components.

Figure 1 summarizes the chronicle-based approach architecture. The system is composed of a global diagnoser (or broker, detailed in section 5.2), in charge of merging the local diagnoses, and a finite number of services, each one being composed of:

- the web service itself,
- logs generated in real time by the web service,
- a base of chronicles, generated off-line,
- a local diagnoser, using the logs to instantiate chronicles from the base.

Figure 1 also shows communications between global and local diagnosers. Communication details are presented in section 5.

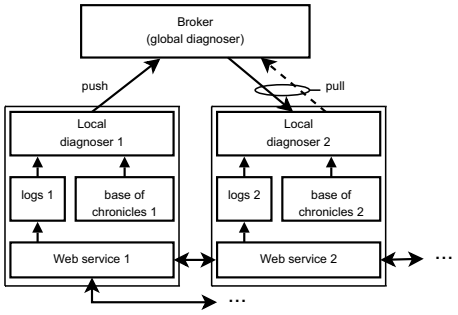


Figure 1: General framework of the distributed chronicle-based approach

In this article, we call "web service" a process described in an orchestration language such as BPEL[1], executed on an appropriate execution engine. A web service will thus be considered as a structured process gathering locally executed activities and calls to remote web services (Figures 2, 3, 4 and 5).

[1]BPEL: Business Process Execution Language

3 Example of the "BN" provider

The test application used in the project deals with an e-shopping service. To illustrate the ideas developed in this article, we use a simplified example that keeps the essential properties of the applications we plan to monitor. We consider *closed* environments, where a workflow-like description of each web service involved in the processing of the request is supposed to be available.

3.1 Presentation

Three web services take part in the BN[2] example: the Eater, the BN provider (BNP) and the unitary provider (BN1). The BNP is, in fact, nothing more than an interface between the producer (BN1) and the consumer (Eater).

For instance, when a hungry Eater asks the BNP for a BN via an $orderBN(1)$ call, the BNP executes $getOneBN()$, claiming a BN to the unitary provider (BN1), before sending it to the Eater. Now, when the Eater asks for n BNs at once (via an $orderBN(n)$ call), the BNP executes n times $getOneBN()$ before returning a result to the Eater.

Figure 2 illustrates the operating principle of these three web services. One can notice that BN1 returns a boolean that denotes if the BN is delivered or not and BNP returns an integer that denotes the number of BNs actually delivered. Indeed, as soon as BN1 stops to deliver BNs, BNP doesn't insist and returns the current number of BNs.

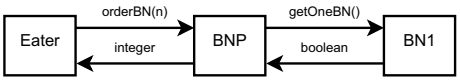


Figure 2: Principle of the exchanges between the services of the BN example

3.2 Faults

We can distinguish two types of faults:

- faults supported by the workflow of the service,
- faults unsupported by the workflow.

Supported faults are handled by the code of the considered web service. They belong to the "normal" behavior of the service. Unsupported faults may provoke an unexpected behavior of the service, or even freeze it.

On the following workflows, faults and resulting timeouts are represented by pentagons.

On the Eater service (Figure 3), the *dataAcquisitionError* is an unsupported fault that induces a number of received BNs differing from the expected one. On this service, we consider that a call to the BNP does not necessarily get a response. That is why we introduce a potential timeout.

The BNP (Figure 4) does not have any inner fault. However, calling BN1, BNP is not sure to get a response. Here again, we introduce a potential timeout.

Three faults are declared on BN1 (Figure 5): the *emptyComputerStock* (the variable reflecting the value of the real

[2]BN is a famous french chocolate biscuit

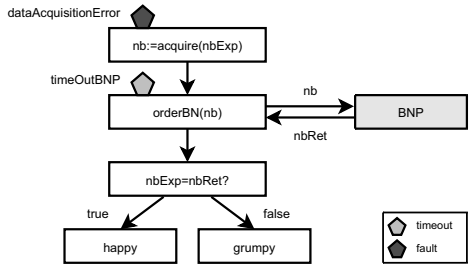


Figure 3: Workflow of the Eater

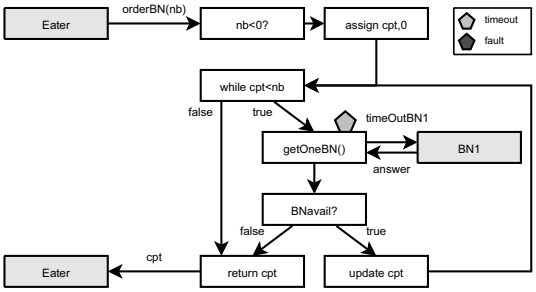


Figure 4: Workflow of the BNP

stock is null), the *hardwareError* and the *emptyRealStock*. The *emptyComputerStock* is a fault supported by the system. In this case, the service replies $false$ to indicate that no BN could be provided. The *hardwareError* and *emptyRealStock* are unsupported faults: the computer stock and the real stock are supposed to be consistent. If one of those two faults occurs, the service freezes, which fires a first timeout on BNP, and a second one on the Eater.

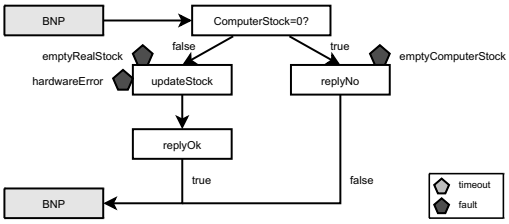


Figure 5: Workflow of the BN1

3.3 Scenarii

We call scenario a complete execution of a set of web services, whatever its status (faulty or not). In the following, we consider two scenarii.

In the first one, the Eater asks the BNP for two BNs (Figure 6.a). After the Eater's request, the BNP executes $getOneBN()$ twice and receives a BN from BN1 twice. This is the ideal case in which the Eater receives what he ordered. In the second scenario, the Eater orders the same number of BNs (Figure 6.b). This time, during the second $getOneBN()$ call, the $emptyComputerStock$ fault occurs on BN1 that does not return a BN. The BNP finally supplies the Eater with only one BN.

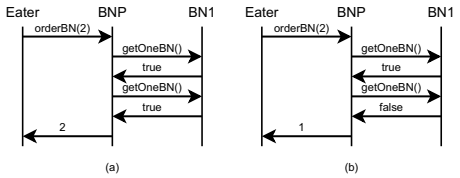


Figure 6: Sequence diagrams corresponding to two scenarios. (a) Everything is alright. (b) A fault on BN1 constrains the Eater to receive only a part of his requested BNs.

4 Representation of distributed chronicles

The nature of the events we have to process on the web services made us choose to base our local diagnosers on chronicle recognition, the formalism of which we recall here. A fault occurring on a service unfortunately often propagates to other services, as explained in the second scenario of section 3.3.

As a consequence, we enrich the initial formalism with synchronization elements that will allow the global diagnoser to spot homologous chronicles and merge them. This operation is detailed in section5.

4.1 Formalism of chronicles *à la* Dousson

In this section we present the formalism of the chronicles, as defined by Dousson (Dousson *et al.* 1993). On each event of the examples, act^- means the beginning of act, act^+ the end of act, and $?var$ means that var is a variable.

Event An *event type* is a term $label(vars)$ where $label$ is an observable emitted during the event and $vars$ the set of variables linked with this event.
Example: $orderBN^-(?nb)$
An *event* is a pair (E, t) where E is an *event type* and t the occurrence date of the event.
Example: $(orderBN^-(?nb), ?t)$
An *event instance* is an event, which variables and date have been instantiated.
Example: $(orderBN^-(?nb = 3), ?t = 2)$
An *event log* $\mathcal{L}$ is a temporally ordered list of event instances.
Example: $(acquire^-(?nb = 3), ?t_1 = 1)$
$(orderBN^-(?nb = 3), ?t_2 = 2)$
$(orderBN^+(?nbReturned = 3), ?t_3 = 5)$
$\dots$

Chronicle A *chronicle model* $\mathcal{C}$ is a pair $(\mathcal{S}, \mathcal{T})$ where $\mathcal{S}$ is a set of events[3] and $\mathcal{T}$ a set of constraints between their occurrence dates.
Example: $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ with

[3]We can add optional events. Then, the chronicle model $\mathcal{C}$ is still a pair $(\mathcal{S}, \mathcal{T})$, but with $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$ with $\mathcal{S}^+$ the set of mandatory events and $\mathcal{S}^-$ the set of optional events of the chronicle.

$\mathcal{S} = \{ \quad (acquire^-(?nb), ?t_1),$
$(orderBN^-(?nb), ?t_2),$
$(orderBN^+(?nbReturned), ?t_3),$
$(nbExpected = nbReturned?^-$
$(?nbReturned), ?t_4),$
$(happy(), ?t_5)$
$\}$ and
$\mathcal{T} = \{?t_1 < ?t_2, ?t_2 < ?t_3, ?t_3 < ?t_4, ?t_4 < ?t_5\}$

A *partially instantiated chronicle* c from a model $\mathcal{C}$ is a set of event instances of $\mathcal{C}$ consistent with the temporal constraints of $\mathcal{C}$.
Example:
$c = \{ \quad (acquire^-(?nb = 3), ?t_1 = 1),$
$(orderBN^-(?nb = 3), ?t_2 = 2),$
$(orderBN^+(?nbReturned = 3), ?t_3 = 5)$
$\}$

A *fully instantiated chronicle* c from a model $\mathcal{C}$ is a set containing an instance of each event of $\mathcal{C}$ and which is consistent with the temporal constraints of $\mathcal{C}$.
Example:
$c = \{ \quad (acquire^-(?nb = 3), ?t_1 = 1),$
$(orderBN^-(?nb = 3), ?t_2 = 2),$
$(orderBN^+(?nbReturned = 3), ?t_3 = 5),$
$(nbExpected = nbReturned?^-$
$(?nbReturned = 3), ?t_4 = 7),$
$(happy(), ?t_5 = 8)$
$\}$

4.2 Extension of the formalism of chronicles

In this section, we extend the previous definitions and define the concept of distributed chronicle.

Event We first enrich the notion of event, in order to allow some events of a chronicle to trigger a global diagnosis stage without having to wait for the full recognition of this chronicle.

A *brokering event* is an event triggering the global diagnoser via a push call without a full recognition of the chronicle.

An *event of a distributed chronicle* is a tuple (E, t, b) representing an event enriched with a boolean b denoting if the event is a *brokering event* (bro) or not ($\neg bro$).

Synchronization point Then, we extend the formalism of chronicles, adding the synchronization elements themselves.

The *set of chronicle variables* of $\mathcal{C}$ is the set of non temporal variables that belong to the events of this chronicle.

A *synchronization variable* is a pair $(var, bool)$ where var is a chronicle variable and *bool* a boolean denoting the variable status inside a given chronicle model: erroneous (err) or not ($\neg err$).
Example: $(?nb, \neg err)$

A *synchronization point* is a tuple $(event, \{vars_c\}, serv_{type})$ where $event$ is an event, $\{vars_c\}$ a set of synchronization variables linked with this event and $serv_{type}$ a type of remote service the local service communicates with.
Example: $((orderBN^-(?nb,), ?t_2, \neg bro),$
$\{(?nb, \neg err)\}, provider)$

An *instance of synchronization point* is a synchronization point in which events and variables are instantiated, and $serv_{type}$ is instantiated as $serv_{id}$, *i.e.* the effective address of the remote service.
Example: $((orderBN^-(nb = 3), ?t_2 = 2, \neg bro),$
$\{(?nb, \neg err)\}, provider = BNP)$

An *incoming/outgoing synchronization point* is an oriented synchronization point. Incoming stands for a $serv_{remote} \rightarrow serv_{local}$ communication, outgoing for the contrary. By extension, an *incoming/outgoing infection point* is an oriented synchronization point having at least one erroneous (err) variable in $\{vars_c\}$.

Distributed chronicle Finally, we introduce several concepts of distributed chronicles, endowing them with a notion of "precaution triggering".

Distributed chronicle: a distributed chronicle is a classical chronicle enriched with a "synchronization" part, so that we could merge it with chronicles from adjacent services. A distributed chronicle $\mathcal{C}_\mathcal{D}$ is a tuple $(\mathcal{S}, \mathcal{T}, \mathcal{O}, \mathcal{I})$ where $\mathcal{S}$ is a set of events, $\mathcal{T}$ a graph of constraints between their occurrence dates, and $\mathcal{O}$ and $\mathcal{I}$ are respectively two sets of outgoing and incoming synchronization points.
Examples: Considering the distributed chronicle of an Eater getting all his BNs (Figure 7.a), we have $\mathcal{C}_{\mathcal{D}1} = (\mathcal{S}_1, \mathcal{T}_1, \mathcal{O}_1, \mathcal{I}_1)$, with
$\mathcal{S}_1 = \{ \quad (acquire^-(?nb), ?t_1, \neg bro),$
$(orderBN^-(?nb), ?t_2, \neg bro),$
$(orderBN^+(?nbReturned), ?t_3, \neg bro),$
$(nbExpected = nbReturned?^-$
$(?nbReturned), ?t_4, \neg bro),$
$(happy(), ?t_5, \neg bro)$
$\}$
$\mathcal{T}_1 = \{?t_1 < ?t_2, ?t_2 < ?t_3, ?t_3 < ?t_4, ?t_4 < ?t_5\}$
$\mathcal{O}_1 = \{((orderBN^-(?nb), ?t_2, \neg bro),$
$\{(?nb, \neg err)\}, provider)\}$
$\mathcal{I}_1 = \{((orderBN^+(?nbReturned), ?t_3, \neg bro),$
$\{(?nbReturned, \neg err)\}, provider)\}$

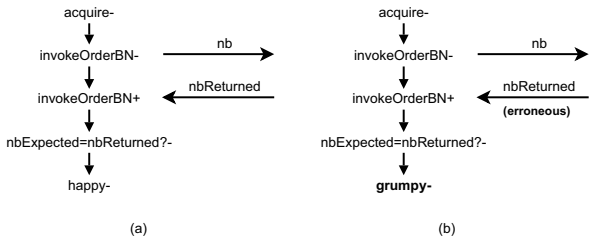


Figure 7: (a) Normal chronicle of the Eater and (b) chronicle of the Eater receiving an unexpected number of BNs

Considering the distributed chronicle for which BNP returns a bad number of BNs (Figure 7.b), we have $\mathcal{C}_{\mathcal{D}2} = (\mathcal{S}_2, \mathcal{T}_2, \mathcal{O}_2, \mathcal{I}_2)$, with

$$
\begin{aligned}
\mathcal{S}_2 = \{ \quad & (acquire^-(?nb), ?t_1, \neg bro), \\
& (orderBN^-(?nb), ?t_2, \neg bro), \\
& (orderBN^+(?nbReturned), ?t_3, \neg bro), \\
& (nbExpected = nbReturned?^- \\
& \qquad (?nbReturned), ?t_4, \neg bro), \\
& (grumpy(), ?t_5, bro) \\
\}& \\
\mathcal{T}_2 = & \{?t_1 < ?t_2, ?t_2 < ?t_3, ?t_3 < ?t_4, ?t_4 < ?t_5\} \\
\mathcal{O}_2 = & \{((orderBN^-(?nb), ?t_2, \neg bro), \\
& \qquad \{(?nb, \neg err)\}, provider)\} \\
\mathcal{I}_2 = & \{((orderBN^+(?nbReturned), ?t_3, \neg bro), \\
& \qquad \{(?nbReturned, err)\}, provider)\}
\end{aligned}
$$

Colored distributed chronicle: by extension, a colored distributed chronicle $\mathcal{C}_{\mathcal{DC}}$ is a tuple $(\mathcal{S}, \mathcal{T}, \mathcal{O}, \mathcal{I}, \mathcal{K})$ where $\mathcal{K}$ is a "color". $\mathcal{K}$ represents the degree of importance of the chronicle. Using three colors (green, orange and red), we can imagine a simple strategy.

- A "normal" execution will be represented by a green chronicle which, even fully recognized, will not trigger the global diagnoser. The chronicle on Figure 7.a could be a green one. Those chronicles will be used by the broker during stages of information gleaning, when there is a need to account for the normal execution of one service.
- An "abnormal" execution will be represented by an orange chronicle which will have to be fully recognized to trigger the broker. The chronicle on Figure 7.b could be an orange one.
- A particularly risky execution will be represented by a red chronicle which will trigger the broker when only $n\%$ recognized, unless a brokering event occurs before.

5 Computation of the distributed diagnosis

As mentioned in section 2, the computation of the global diagnosis relies on local diagnosers, one per service, and on a global diagnoser merging the local diagnoses (Figure 1).

5.1 Local diagnoser

Several information are required by the local diagnoser to operate: first, a base of chronicles covering all the possible execution cases of the service, as explained in the previous section; then, an event log fed in real time (flow of events) and allowing the chronicle recognition system to instantiate the chronicle models previously created; last, an evaluator that polls the evolution of the chronicles inside the chronicle recognition system, so that it can trigger the global diagnoser if needed (see section 5.2). The evaluator can also be used by the global diagnoser to prune chronicles that are inconsistent with the global diagnosis. This allows to refine the local diagnosis thanks to global information.
The general framework of a local diagnoser is illustrated in Figure 8.a. The local diagnosers are based on a chronicle recognition system called CRS, developed by Dousson (Dousson 1994).

5.2 Global diagnoser: the broker

The global diagnoser (see Figure 8.b), also called broker, is in charge of synchronizing the local diagnoses. One can note

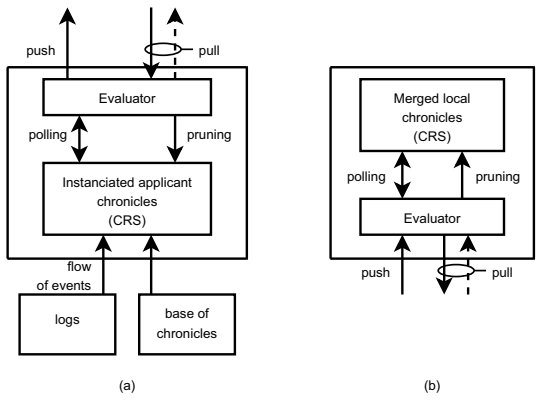


Figure 8: Working of the (a) local and (b) global diagnosers

two operating modes between local and global diagnosers:

- the *push* or event-based mode,
- the *pull* or query-based mode.

Those modes are jointly used to merge local diagnoses in order to compute a global diagnosis of the studied system.

The *push* mode is used by local diagnosers to trigger a global diagnosis stage or to inform the broker of important events when a global diagnosis stage is already running. The distributed chronicles that may trigger the broker are:

- a non-green chronicle when fully recognized,
- a red chronicle when recognized at $n\%$,
- a chronicle in which a brokering event has been recognized.

The *pull* mode is used by the broker to ask the local diagnosers for additional information, and merge their diagnoses.

The global diagnoser, triggered by a *push* call from a local diagnoser, will then issue a series of *pull* calls. Nevertheless, it can take advantage of opportune *push* calls to refine its diagnosis. In order to enforce the update of its information, the *broker* can also use a kind of polling to question regularly the services contained in the set S of services concerned by the execution.

A meta-algorithm describing the information update operation of the global diagnoser is given in Figure 9. This meta-algorithm could be improved, especially by taking into account synchronous services for which call and return synchronization points could be dealt conjointly.

5.3 Computation of the global diagnosis

The diagnosis merging algorithm is currently being implemented and will be validated on the BN example, presented in section 3. The principle of this algorithm is presented on a simple example, not taking into account the triggering policy we presented before, but self-explanatory enough. This example is extracted from the BN example, Figure 10 illustrates the considered services and synchronization variables.

```
/* on the broker */
//set of services taking part in the choreography
S = ∅
//sync constraints known by the broker
C_glob = ∅
//call of the broker by a service s
procedure broker::push (service s) {
        S = S ∪ {s}
        broker::pull(s)
}
//query of a service s by the broker
procedure broker::pull (service s) {
        ask s for the merged constraints c of the candidate
                chronicles on its sync points P
        update C_glob thanks to c
        foreach sync point p from P do
                if c modified the knowledge on p then
                        c_r = known constraints on p sync vars
                        s_r = partner service linked with s by p
                        c' = s_r.service::pruneChronicles(c_r)
                        update C_glob thanks to c'
                        broker::push(s_r)
                fi
        done
}
/* on the services */
//murder of chronicles not verifying c
procedure service::pruneChronicles (constraints c) {
        kill chronicle instances not verifying c
        compute constraints c' on the sync vars
        return c'
}
```

Figure 9: Algorithm of information update for the broker

On the Eater service, we consider three chronicles (see Table 1). The first one represents the ideal execution case, the second one represents the *dataAcquisitionError*, the third one represents an external error resulting in a wrong number of supplied BNs.

On the BNP, we also consider three chronicles. The first one is the ideal case, the second one is the case when BNP receives an erroneous request and returns the requested number of BNs, the third one represents an external error preventing BNP from receiving all the BNs he ordered.

On the BN1 at last, there are two chronicles. The first one is the ideal case, the second one represents an *emptyComputerStock*.

In order to restrain the number of calls to the broker, we consider that the algorithm already provides the conjunction of synchronization points in the case of synchronous services. Thus, the broker will execute the *pruneChronicles()* method only once instead of twice.

At the beginning of the execution, all the chronicles are candidate. Let us assume that the Eater calls the broker after having recognized two local chronicles that have the same signature (sequence of events), *dataAcqErr* and *extErr*. There are two candidate chronicles on this service:

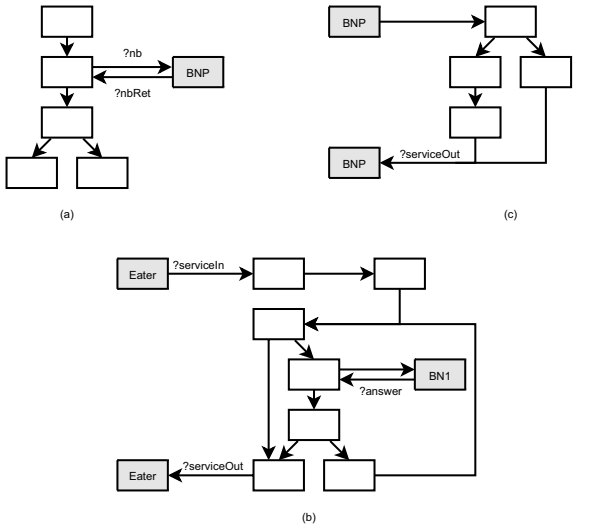


Figure 10: Abstract workflow and synchronization variables of (a) the Eater, (b) the BNP and (c) the BN1

- *dataAcqErr* with $?nb$ being err and $?nbRet$ being err,
- *extErr* with $?nb$ being $\neg err$ and $?nbRet$ being err.

In those two chronicles, one can note that $?nbRet$ is err whereas nothing can be said about $?nb$. In order to differentiate those chronicles, the broker queries BNP, the partner service that shared $?nb$ and $?nbRet$ with the Eater. $?nb$ and $?nbRet$ on the Eater correspond respectively to $?serviceIn$ and $?serviceOut$ on BNP. Then, the broker asks BNP for its recognized chronicles having an err $?serviceOut$ and an indifferently err or $\neg err$ $?serviceIn$. Two chronicles are candidate:

- *fwdErr* with $?serviceOut$ being err, $?serviceIn$ being err and $?answer$ being $\neg err$,
- *extErr* with $?serviceOut$ being err, $?serviceIn$ being $\neg err$ and $?answer$ being err.

Here again, the broker interrogates the partner service of BNP that transmitted the $?answer$ variable, *i.e.* BN1. Chronicles *fwdErr* and *extErr* having an $?answer$ indifferently err or $\neg err$, the broker asks BN1 for its recognized chronicles having a $?serviceOut$ indifferently err or $\neg err$. Assuming BN1 answers that *emptyCptStock* has been recognized, the broker has enough information to refine the local diagnoses of services BNP and Eater.

Chronicle *emptyCptStock* on BN1 having an err $?serviceOut$, only chronicles with an err $?answer$ (*i.e. extErr*) are kept on BNP. So, we know exactly what happened on BNP ($\neg err$ $?serviceIn$ and err $?serviceOut$). By propagating these information to the Eater, chronicle *dataAcqErr* is killed and only *extErr* is kept. We know exactly what happened on the Eater.

Finally, thanks to the err status of $?serviceOut$ in the

Eater				
Variable	Partner	Ideal case	dataAcqErr	extErr
$?nb$	BNP	$\neg err$	err	$\neg err$
$?nbRet$	BNP	$\neg err$	err	err

BNP				
Variable	Partner	Ideal case	fwdErr	extErr
$?serviceIn$	$Eater$	$\neg err$	err	$\neg err$
$?answer$	$BN1$	$\neg err$	$\neg err$	err
$?serviceOut$	$Eater$	$\neg err$	err	err

BN1			
Variable	Partner	Ideal case	emptyCptStock
$?serviceIn$	BNP	$\neg err$	$\neg err$
$?serviceOut$	BNP	$\neg err$	err

Table 1: Chronicles for Eater, BNP and BN1, with synchronization variables status

chronicle recognized on BN1, the broker is able to refine the local diagnoses of BNP, first, and of the Eater, then. At the end of the global diagnosis process, we know that neither the Eater nor the BNP committed errors.

6 Related work and discussion

Within the context of the supervision of dynamic systems, many works use the formalism of chronicles (Cordier & Dousson 2000; Cordier *et al.* 1998; Dojat *et al.* 1998; Quiniou *et al.* 2001; Aguilar *et al.* 1994). Nevertheless, few deal with using chronicles in a distributed context. The approach presented in (Boufaied *et al.* 2004) focuses on temporal aspects and proposes a distributed checking of temporal constraints (by introducing both local and global temporal constraints). In (Guerraz & Dousson 2004), the authors study the problem of acquiring chronicles from the fault model of a system, described with Petri nets. They use a method based on unfolding Petri nets. The formalism of chronicles is enriched with pre- and post-conditions on the current system state, and the recognition algorithm modified consequently. However, to our knowledge, nobody directly worked on the use of distributed chronicles, in particular on the integration of synchronization constraints between components inside the formalism and on the adaptation of the corresponding algorithm, as we propose in this article.

The way we approach the problem of monitoring dynamic systems from a distributed-chronicle-based modeling of the system may be compared with works dealing with distributed approaches of monitoring discrete-event systems, such as (Baroni *et al.* 2000; Debouk *et al.* 2000; Aghasaryan *et al.* 1998; Jiroveanu & Boël 2005; Pencolé & Cordier 2005; Roos *et al.* 2002; Provan 2002). In each of those works, local diagnoses computed by the different components of the system are synchronized in order to compute a diagnosis taking into account the constraints between components. For instance, the approach of (Aghasaryan *et al.* 1998) is not so far away from ours, as they fit parts together to build the system diagnosis, like in a puzzle. Those parts, called *tiles*, are labelled by alarms and represent pieces of trajectories. The main difference between the two approaches, apart from the Petri-net-based formalism they use, is that theirs is fully distributed and uses communications between local components to do the computations, without any supervisor. In our decentralized case, a supervisor is in charge of fitting local chronicles together after having synchronized them, so that a global chronicle could be built.

(Grosclaude 2004) is also interested in software components monitoring. The components of the system are described by Petri nets and each component is associated with a local controller that monitors the evolution of this component, observing the messages exchanged between the component and its neighbors.

Concerning web services monitoring, we can cite (Yan *et al.* 2005), the objective of which is to acquire a model as automata that will permit to monitor components thanks to the BPEL description of their process. Closer to us, (Lazovik *et al.* 2003) proposes to use planning tools to allow the user to express his requests thanks to a high-level language and to control the execution of his plans by interlacing execution and plan update. The authors of (Baresi *et al.* 2004) are interested in checking on line the consistency between what a web service should do, called a contract, and its effective execution. Contracts are expressed as constraints in a constraint-oriented language, and integrated in the BPEL files under the shape of annotations. Then, monitors, implemented as web services, observe the behavior of the web services and are capable of detecting timeout problems or functional errors. In (Barbon *et al.* 2006), a quite similar approach relies on a monitoring of plans to monitor requests and uses the KPLTL temporal logic in order to express the specifications that have to be respected.

As a matter of fact, the decentralized architecture we propose is totally coherent with the approach proposed in (Ardissono *et al.* 2005). This approach is also linked with the WS-DIAMOND project. Here, each web service is equipped with a local diagnoser generating hypotheses that are consistent with the local model and the observations. A supervisor merges local diagnoses to compute a global one, by propagating hypotheses from a local diagnoser to its neighbors. The main difference is that they rely on a static diagnosis approach: using dependencies between state variables, their approach consists of explaining the alarms that arise at a given time. In our case, we monitor the behavior of the components as it evolves. This allows, on the one hand, to identify problems related to alarm firing and, on the other hand, to forestall a potential problem and avoid its occurrence. This vision also allows us to consider a monitoring of the quality of service offered by the different web services.

7 Conclusion

We showed, in this article, how to extend the formalism of chronicles to decentralized diagnosis. In particular, the introduction of "synchronization points" allows the global diagnoser to merge local diagnoses. Our goal, in short, is to test our approach on a more realistic example, the e-shopping one, adopted by most of WS-DIAMOND partners.

A classical problem, using chronicle-based methods, is the chronicle acquisition. For the BN example, we built chronicles ourselves from the workflows of the different services. The idea was to build "maximal" chronicles, which means we included all the occurring events inside. It would be worthwhile, taking inspiration from (Guerraz & Dousson 2004; Yan *et al.* 2005), to build chronicles from workflows automatically and, in order to simplify the resulting chronicles, to write chronicle minimization algorithms, algorithms that would keep discriminating events only.

References

Aghasaryan, A.; Fabre, E.; Benveniste, A.; Boubour, R.; and Jard, C. 1998. Fault detection and diagnosis in distributed systems : an approach by partially stochastic petri nets. *Discrete Event Dynamic Systems* 8(2):203–231.

Aguilar, J.; Bousson, K.; Dousson, C.; Ghallab, M.; Guasch, A.; Milne, R.; Nicol, C.; Quevedo, J.; and Travé-Massuyès, L. 1994. Tiger: real-time situation assessment of dynamic systems. Technical Report 94445, LAAS.

Ardissono, L.; Console, L.; Goy, A.; Petrone, G.; Picardi, C.; Segnan, M.; and Theseider Dupré, D. 2005. Cooperative model-based diagnosis of web services. In *Proc. of the 16th Int. Workshop on Principles of Diagnosis (DX'05)*.

Barbon, F.; Traverso, P.; Pistore, M.; and Trainotti, M. 2006. Run-time monitoring of instances and classes of web service compositions. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS'06)*, 63–71.

Baresi, L.; Ghezzi, C.; and Guinea, S. 2004. Smart monitors for composed services. In *Proc. of the 2nd Int. Conf. on Service-Oriented Computing (ICSOC'04)*, 193–202.

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 2000. Diagnosis of a class of distributed discrete-event systems. *IEEE Transactions on systems, man, and cybernetics* 30(6):731–752.

Boufaied, A.; Subias, A.; and Combaceau, M. 2004. Distributed fault detection with delays consideration. In *Proc. of the 15th Int. Workshop on Principles of Diagnosis (DX'04)*, 57–62.

Cordier, M.-O., and Dousson, C. 2000. Alarm driven monitoring based on chronicles. In *Proc. of Safeprocess'2000*, 286–291.

Cordier, M.-O.; Krivine, J.; Laborie, P.; and Thiébaux, S. 1998. Alarm processing and reconfiguration in power distribution systems. In *Proc. of IEA-AIE'98*, 230–240.

Debouk, R.; Lafortune, S.; and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems* 10(1-2):33–86.

Dojat, M.; Ramaux, N.; and Fontaine, D. 1998. Scenario recognition for temporal reasoning in medical domains. *Artificial Intelligence in Medicine* 14(1-2):139–155.

Dousson, C.; Gaborit, P.; and Ghallab, M. 1993. Situation recognition: representation and algorithms. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI'93)*, 166–172.

Dousson, C. 1994. Chronicle Recognition System. http://crs.elibel.tm.fr/.

Grosclaude, I. 2004. Model-based monitoring of component-based software systems. In *Proc. of the 15th Int. Workshop on Principles of Diagnosis (DX'04)*, 51–56.

Guerraz, B., and Dousson, C. 2004. Chronicles construction starting from the fault model of the system to diagnose. In *Proc. of the 15th Int. Workshop on Principles of Diagnosis (DX'04)*, 51–56.

Jiroveanu, G., and Boël, R. 2005. Petri net model-based distributed diagnosis for large interacting systems. In *Proc. of the 16th Int. Workshop on Principles of Diagnosis (DX'05)*, 25–30.

Lazovik, A.; Aiello, M.; and Papazoglou, M. 2003. Planning and monitoring the execution of web service requests. In *Proc. of the 1st Int. Conf. on Service-Oriented Computing (ICSOC'03)*, volume 2910 of *Lecture Notes in Computer Science*, 335–350.

Pencolé, Y., and Cordier, M.-O. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence Journal* 164(1-2):121–170.

Pencolé, Y.; Cordier, M.-O.; and Rozé, L. 2002. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *IEEE Conf. on Decision and Control (CDC'02)*, 435–440.

Provan, G. 2002. A model-based diagnosis framework for distributed systems. In *Proc. of the 13th Int. Workshop on Principles of Diagnosis (DX'02)*, 16–25.

Quiniou, R.; Cordier, M.-O.; Carrault, G.; and Wang, F. 2001. Application of ilp to cardiac arrhythmia characterization for chronicle recognition. In *ILP'2001*, volume 2157 of *LNAI*, 220–227.

Roos, N.; Teije, A.; Bos, A.; and Witteveen, C. 2002. An analysis of multi-agent diagnosis. In *Proc. of the 1st Int. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS'02)*.

Salaün, G.; Bordeaux, L.; and Schaerf, M. 2004. Describing and reasoning on web services using process algebra. In *Proc. of the IEEE Int. Conf. on Web Services (ICWS'04)*, 43.

Yan, Y.; Pencolé, Y.; Cordier, M.-O.; and Grastien, A. 2005. Monitoring web service networks in a model-based approach. In *3rd European Conf. on Web Services (ECOWS)*.

Diagnosing Intermittent Faults

Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
dekleer@parc.com

Abstract

Almost all approaches to model-based diagnosis presume that the system being diagnosed behaves non-intermittently and analyze this behavior over a small number (often one) of time instants. In this paper we show how existing approaches to model-based diagnosis can be extended to diagnose intermittent failures as they manifest themselves over numbers of time instants. In addition, we show where to insert probe points to best distinguish among the intermittent faults those that best explain the symptoms and isolate the fault in minimum expected cost.

1 Introduction

Experience with diagnosis of automotive systems and reprographic machines [Fromherz *et al.*, 2003] shows that intermittent faults are among the most challenging kinds of faults to isolate. Such systems raise many modeling complexities, so we present our approach to isolating intermittent faults in the context of logic systems. For benchmarks, we draw on the circuits in [Brglez and Fujiwara, 1985].

The notion of intermittency is a hard-to-define concept, so we first describe it intuitively before defining it more formally. A system consists of a set of components. A faulty component is one which is physically degraded such that it will not always function correctly. For example, a faulted resistor may no longer conduct the expected current when a specified voltage is applied across it. A worn roller in a printer may no longer be able to grip the paper consistently thereby causing intermittent paper jams. In the case of a worn roller, it usually operates correctly but will infrequently slip and cause a paper jam. We therefore associate two probabilities with each component: (1) the probability that the actual component deviates from its design such that it may exhibit a malfunction, and (2) the conditional probability that the faulted component malfunctions when observed. For example, the probability of a roller being worn might be 10^{-5} while the probability of a worn roller actually malfunctioning might be .01.

We do not model the dynamic behavior of a system over the time. Instead, the system is viewed in a sequence of observation events. For example, a piece of paper is fed into a printer, it may be observed to jam. A test-vector is applied to a combinational digital circuit and its output signals subsequently observed.

Definition 1 *An intermittently faulted component is one whose output(s) are not a function of its inputs.*

Intermittency can arise from at least two sources. If the system can be modeled at a more detailed level, apparent intermittency can disappear. For example, two wires may be shorted, making it look like a gate is intermittent, when in fact there is an unmodeled and unwanted connection. The second type of intermittency arises from some stochastic physical process which is intermittent at any level of detail. This paper focuses on this second type of intermittency.

Diagnosis of intermittent faults is a broad challenge. In this paper we make the following presuppositions:

- The device contains only one intermittent fault.
- No non-intermittent fault. Components have deterministic outputs.
- One observed variable per time instant.
- All inputs are always known, but may change over time.
- Diagnosis starts with an observation in which a faulty output is observed.
- No memory or energy storage components.
- No measurement error.

None of these limitations present a fundamental obstacle to our model-based approach, but are topics for future research.

2 GDE probability framework

This basic framework is described in [de Kleer and Williams, 1987; de Kleer *et al.*, 1992].

Definition 2 *A system is a triple (SD,COMPS, OBS) where:*

1. *SD, the system description, is a set of first-order sentences.*
2. *COMPS, the system components, is a finite set of constants.*
3. *OBS, a set of observations, is a set of first-order sentences.*

Definition 3 *Given two sets of components Cp and Cn define $\mathcal{D}(Cp, Cn)$ to be the conjunction:*

$$\Big[\bigwedge_{c\in Cp} AB(c)\Big] \wedge \Big[\bigwedge_{c\in Cn} \neg AB(c)\Big].$$

Where $AB(x)$ represents that the component x is ABnormal (faulted).

A diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 4 *Let $\Delta \subseteq COMPS$. A diagnosis for (SD,COMPS,OBS) is $\mathcal{D}(\Delta, COMPS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \{\mathcal{D}(\Delta, COMPS - \Delta)\}$$

Definition 5 *An AB-literal is $AB(c)$ or $\neg AB(c)$ for some $c \in COMPS$.*

Definition 6 *An AB-clause is a disjunction of AB-literals containing no complementary pair of AB-literals.*

Definition 7 *A conflict of (SD,COMPS,OBS) is an AB-clause entailed by $SD \cup OBS$.*

2.1 Representing time

Time is expressed easily in the preceding formalism. For example, the model of an inverter is often written as:

$$INVERTER(x) \rightarrow \Big[\neg AB(x) \rightarrow [in(x,t) = 0 \equiv out(x,t) = 1]\Big].$$

When ambiguous this paper represents the value v of variable x at time t as $T(x = v, t)$. Time is a sequence of instants t_0, t_1, ... The probability of X at time t is represented as $p_t(X)$.

2.2 Updating diagnosis probabilities

Components are assumed to fail independently. Therefore, the prior probability a particular diagnosis $\mathcal{D}(Cp, Cn)$ is correct is:

$$p_1(\mathcal{D}) = \prod_{c\in C_p} p(c) \prod_{c\in C_n} (1 - p(c)), \quad (1)$$

where $p(c)$ is the prior probability that component c is faulted.

The posterior probability of a diagnosis $\mathcal{D}$ after an observation that x has value v at time t is given by Bayes Rule:

$$p_t(\mathcal{D}|x = v)) = \frac{p_t(x = v|\mathcal{D})p_{t-1}(\mathcal{D})}{p_t(x = v)}. \quad (2)$$

$p_{t-1}(\mathcal{D})$ is determined by the preceding measurements and prior probabilities of failure. The denominator $p_t(x = v)$ is a normalizing term that is identical for all $p(\mathcal{D})$ and thus needs not be computed directly. The only term remaining to be evaluated in the equation is $p_t(x = v|\mathcal{D})$:

$p_t(x = v|\mathcal{D}) = 0$ if $\mathcal{D}, SD, OBS, T(x = v, t)$ are inconsistent,

else,

$p_t(x = v|\mathcal{D}) = 1$ if $T(x = v, t)$ follows from $\mathcal{D}, SD, OBS$

If neither holds,

$$p_t(x_i = v_k|\mathcal{D}) = \epsilon_{ik}. \quad (3)$$

Various ϵ-policies are possible [de Kleer, 2006] and a different ϵ can be chosen for each variable x_i and value v_k. Typically, $\epsilon_{ik} = \frac{1}{m}$. This corresponds to the intuition that if x ranges over m possible values, then each possible value is equally likely. In digital circuits $m = 2$ and thus $\epsilon = .5$.

In the conventional framework, observations that differ with predictions yield conflicts, which are then used to compute diagnoses. Consider the simple adder digital circuit of Figure 1. Suppose all the inputs to the circuit are 0, and c_o is measured to be 1. This yields one minimal conflict:

$$AB(A1) \vee AB(A2) \vee AB(O1).$$

And the singlefault diagnoses will thus be: $\mathcal{D}(\{O1\}, \{A1, A2, X1, X2\})$, $\mathcal{D}(\{A1\}, \{O1, A2, X1, X2\})$, and $\mathcal{D}(\{A2\}, \{A1, O1, X1, X2\})$.

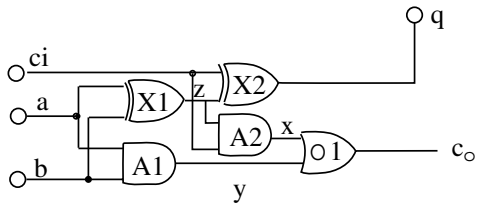


Figure 1: Full adder. This circuit computes the binary sum of c_i (carry in), a and b; q is the least significant bit of the result and c_o (carry out) the high order bit.

3 Extensions to the conventional framework to support intermittent faults

In the conventional framework, $p(c)$ is the prior probability that component c is faulted. In the new framework, two probabilities are associated with each component: (1) $p(c)$ represents the prior probability that a component is intermittently faulted, and (2) $g(c)$ represents the conditional probability that component c is behaving correctly when it is faulted.

In the intermittent case, the same sequential diagnosis model and Bayes rule update applies. However, a more sophisticated ϵ-policy is needed. Note that an ϵ-policy applies only when a particular diagnosis neither predicts $x_i = v_k$ nor is inconsistent with it. Consider the case where there is only a singlefault. As we assume all inputs are given, the only reason that a diagnosis could not predict a value for x_i is when the faulted component causally affects x_i. Consider the singlefault diagnosis $\mathcal{D}(\{c\}, COMPS - \{c\})$. If c is faulted, then $g(c)$ is the probability that it is producing a correct output. There are only two possible cases: (1) c is outputting a correct value with probability $g(c)$, (2) c is outputting a faulty value with probability $1 - g(c)$. Therefore, if $x_i = v_k$ follows from $\mathcal{D}(\{\}, COMPS), SD, OBS$ and ignoring conflicts:

$$p_t(x_i = v_k|\mathcal{D}(\{c\}, COMPS - \{c\})) = g(c).$$

Otherwise,

$$p_t(x_i = v_k|\mathcal{D}(\{c\}, COMPS - \{c\})) = 1 - g(c).$$

These two ϵ equations are equivalent to (no need to ignore conflicts): $p_t(x_i = v_k|\mathcal{D}) =$

$$\begin{array}{ll} g(c) & \text{if } \mathcal{D}, o(c), OBS, SD \vdash T(x_i = v_k, t) \\ + & \\ 1 - g(c) & \text{if } \mathcal{D}, \overline{o}(c), OBS, SD \vdash T(x_i = v_k, t) \end{array}$$

Here, ($\mathcal{D}$ represents $\mathcal{D}(\{c\}, COMPS - \{c\})$), $o(c)$ is the correct model for component c, and $\overline{o}(c)$ the incorrect (output negated) model.

For example, observing $c_o = 1$ in Figure 1 results 3 singlefaults: $\{[O1], [A1], [A2]\}$. (Purely for brevity, in the rest of the paper we notate $\mathcal{D}(\{c\}, COMPS - \{c\}))$ by $[c]$.) If all components are equally likely to fail a priori, then $p([A1]), p([A2]), p([O1]) = \frac{1}{3}$. Suppose we observe $c_0 = 0$ next. In the conventional framework this has no consequence, as the observation is the same as the prediction. However, in the intermittent case, such 'good' observations provide significant diagnostic information. Table 1 illustrates the results if c_o is first observed to be faulty, $g(c) = .9$ for all components, $O1$ is the actual fault, and y is observed continuously. As sampling continues, $p([A1])$ will continue to drop. An intelligent probing strategy would switch to observing x at time 4. As y is insensitive to any error at $[O1]$, no erroneous value would ever be observed.

It is important to note that the singlefault assumption does not require an additional inference rule. The Bayes rule update equation does all the necessary work. For example, when $c_o = 1$ is observed, the singlefault diagnoses $[X1]$ and $[X2]$ both predict $c_o = 0$ and thus are both eliminated from consideration by the update equation. As an implementation detail, singlefaults can be computed efficiently as the intersection of all conflicts found in all the samples.

4 A simplistic probing strategy for singlefaults

Consider the example of Figure 1 and Table 3. Repeatedly measuring $y = 0$ will monotonically drive down the posterior probability of $A1$ being faulted. Choosing probes which drive down the posterior probability of component faults is at the heart of effective isolation of intermittent faults.

Table 1 suggests a very simple probing strategy from which we can compute an upper bound on the expected number of samples needed to isolate the intermittent component. The simplest strategy is to pick the lowest posterior probability component and repeatedly measure its output until either a fault is observed or its posterior probability drops below some desired threshold.

The number of samples needed depends on the acceptable misdiagnosis threshold e. Diagnosis stops when one diagnosis has been found with posterior probability $p > 1 - e$. To compute the upper bound we: (1) we take no advantage of the internal structure of the circuit; (2) we presume that every measurement can exonerate only one component; (3) we are maximally unlucky and never witness another incorrect output.

Table 1: Probabilities of component failure over time. Row 0 are the prior probabilities of the components intermittently failing, row 1 are the probabilities conditioned on knowing the system has a fault, and row 2 are the probabilities conditioned on $c_o = 1$ and so on.

i	A1	A2	O1	X1	X2	obs
0	10^{-10}	10^{-10}	10^{-10}	10^{-10}	10^{-10}	
1	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	
2	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0	0	$c_o = 1$
3	0.310	0.345	0.345	0	0	$y = 0$
4	0.288	0.356	0.356	0	0	$y = 0$
5	0.267	0.366	0.366	0	0	$y = 0$
6	0.247	0.376	0.376	0	0	$y = 0$
7	0.228	0.386	0.386	0	0	$y = 0$
8	0.209	0.395	0.395	0	0	$y = 0$
9	0.192	0.403	0.403	0	0	$y = 0$
10	0.177	0.411	0.411	0	0	$y = 0$
11	0.162	0.419	0.419	0	0	$y = 0$
12	0.148	0.426	0.426	0	0	$y = 0$

Let n be the number of components c_i, $p_1([c_i])$ derived from the priors, and o_i the number of samples of the output of c_i. To shorten the mathematics we write Bayes rule as:

$$p_t([c_i]) = \alpha p_t(x = v|[c_i])p_{t-1}([c_i]),$$

where, α is chosen such that the posterior probabilities sum to 1. Notice that $p_t(x = v|[c_i]) = g(c_i)$ when the output of component c_i is observed. After t samples:

$$p_t([c_i]) = \alpha g(c_i)^{o_i} p_1([c_i]),$$

where,

$$\Sigma o_i = t,$$

and,

$$\alpha = \frac{1}{\Sigma g(c_i)^{o_i} p_1([c_i])}.$$

Splitting the misdiagnosis threshold evenly across all components, we want to pick o_i such that:

$$\alpha g(c_i)^{o_i} p_1(c_i) < \frac{e}{n}.$$

We need to solve for o_i in:

$$\frac{g(c_i)^{o_i} p_1([c_i])}{\alpha} = \frac{e}{n}.$$

If the o_i are sufficiently large, then $\alpha \approx 1$ so we can solve for o_i in:

$$g(c_i)^{o_i} p_1([c_i]) = \frac{e}{n},$$

$$o_i = \frac{\log e - \log n - \log p_1([c_i])}{\log g(c_i)}.$$

In the case of the full adder example all priors are equal, $g = .9$, and $e = .1$. Hence, a worst case strategy is to measure each point 22 times. As there are only 4 informative probe points, the upper bound on expected cost is 88.

5 Learning $g(c)$

Although the prior probability of component failure can be estimated by the manufacturer or previous experience with that component in similar systems, $g(c)$ typically varies widely. Therefore, we try to learn the $g(c)$ over the diagnostic session instead of initially presuming it has some spectific value.

Estimating $g(c)$ requires significant additional machinery which we only describe in the singlefault case. We estimate $g(c)$ by counting the number of samples c is observed to be functioning correctly or incorrectly. We define $G(c)$ to be the number of times c has been observed working correctly, and $B(c)$ as the number of times c has been observed working incorrectly. Corroboratory measurements which c cannot influence are ignored (conflicting measurements exonerate any component which cannot influence it, in which case $g(c)$ is no longer relevant). We estimate $g(c)$:

$$g(c) = \frac{G(c)}{G(c) + B(c)}.$$

The situations in which c is working incorrectly are straightforward to detect — they are simply the situations in which the Bayes update equation utilizes $p_t(x_i = v_k|\mathcal{D}) = 1 - g(c)$. The cases in which c is working correctly requires additional inferential machinery. Consider again the example of Figure 1 where all the inputs are 0, and the expected $c_o = 0$ is observed. Or-gate $O1$ cannot be behaving improperly because its inputs are both 0, and its observed output is 0. And-gate $A1$ cannot be behaving improperly because its inputs are both 0, and its output must be 0, as $O1$ is behaving correctly and its output was observed to be 0. Analogously, and-gate $A2$ cannot be faulted. However, we have no evidence as to whether $X1$ is behaving improperly or not, because if $X1$ were behaving improperly, its output would be 1, but that cannot affect the observation because and-gate $A2$'s other input is 0. $X2$ cannot causally affect c_o. To summarize, if we observe $c_o = 0$ at an instant in which all the inputs are 0, we have learned that $A1$, $A2$ and $O1$ cannot be misbehaving alone, and we learn nothing about the faultedness of $X1$ and $X2$. Hence, $G(A1)$, $G(A2)$ and $G(O1)$ are incremented. All other counters are left unchanged. As a consequence of observing $c_o = 0$, the $G(A1)$, $G(A2)$, and $G(O1)$ is incremented. The net consequence will be that the diagnostician may have to take (slightly) more samples to eliminate $A1$, $A2$ or $O1$ as faulted.

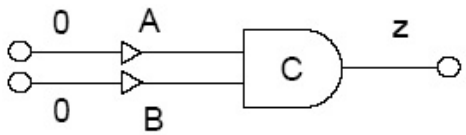


Figure 2: A three component circuit that computes the conjunction of its inputs.

Consider the example of Figure 2. At $t = 0$, the inputs are both 0, and the output is observed to be $z = 0$. Neither the singlefault A nor B can influence $z = 0$, therefore the counters for A and B are left unchanged. However, C can influence the output and therefore $G(C)$ is incremented.

More formally, in response to an observation $x_i = v_k$, $G(c)$ is incremented if a different value for x_i logically follows from the negated model $\overline{o}(c)$; c causally influences the observation and $G(C)$ is incremented.

6 HTMS implementation considerations

The approach has been fully implemented on GDE/HTMS architecture [de Kleer and Williams, 1987; de Kleer, 1992]. Section 10 reports the performance of the approach on various benchmarks.

The Bayes rule update equation requires two repeated checks:

$$\mathcal{D}, o(c), OBS, SD \vdash T(x_i = v_k, t),$$

$$\mathcal{D}, \overline{o}(c), OBS, SD \vdash T(x_i = v_k, t).$$

In the GDE approach each conflict is represented by a positive clause of ATMS assumptions corresponding to the AB-literals. Unfortunately, this representation of conflicts makes it difficult to evaluate these two checks through efficient ATMS operations. The full adder example illustrates the problem very simply (all inputs 0, $c_o = 1$ observed once). If the conflict,

$$AB(A1) \vee AB(A2) \vee AB(O1),$$

were represented directly as an ATMS clause, then no value for c_o could be inferred at any later samples. If the $g(c)$'s were different for the 3 gates it can be useful to measure c_o. In our implementation, every GDE conflict is represented as an ATMS clause which includes the time instant(s) at which it was noticed. The conflict for Figure 1 is represented:

$$(t \neq 2) \vee AB(A1) \vee AB(A2) \vee AB(O1).$$

The Bayes rule update checks are performed within the ATMS which respects the time assumptions.

For the purpose of generating candidates these time assumptions are invisible to GDE. The Bayes rule checks are performed within the ATMS which respects the time assumptions. In this way, the same efficient GDE/Sherlock mechanisms can be exploited for diagnosing intermittent faults. Hence, the ATMS representation of a conflict specifies that at least one of its AB-literals indicates a component which is producing a faulty output at time instant i.

Likewise every inference includes a time assumption. GDE represents prime implicates of the form:

$$(t \neq i) \vee AB(c_1) \vee \cdots \vee AB(c_n) \vee x = v,$$

as:

$$\langle x = v, \{\{(t = i), \neg AB(c_1), \ldots, \neg AB(c_n)\}\}\rangle.$$

The update tests required by both Bayes rule update and the checks to decide whether $G(c)$ should be incremented can often be signficantly sped up. Consider the case where there is only one prime implicate of this form (except for the time assumption) for $x = v$. Given that there is no smaller set of

$\neg AB$'s which support $x = v$, and that an AB component necessarily behaves incorrectly under its $1 - g$ case, it must be the situation that if $x = v$ then any AB changed from negative to positive will result in $x \neq v$. In our example, there is exactly one prime implicate which supports $c_o = 0$:

$$(t = 0) \vee AB(A1) \vee AB(A2) \vee AB(O1) \vee c_o = 0.$$

Therefore, the singlefaults $[O1]$, $[A1]$ and $[A2]$ cannot explain the observation. When there is more than one such prime implicate, we must intersect all the antecedent AB sets. This is similar to a Clark-completion inference [Clark, 1978].

Consider the example of Figure 2 where $z = 0$ is observed. GDE finds two prime implicates which support $z = 0$ (written here as implications):

$$(t = 0) \wedge \neg AB(A) \wedge \neg AB(C) \rightarrow z = 0$$

$$(t = 0) \wedge \neg AB(B) \wedge \neg AB(C) \rightarrow z = 0$$

Under the singlefault assumption, the intersection of antecedents of the implications must be behaving correctly. Therefore, $G(C)$ is incremented.

7 Diagnostic cost

The expected cost of isolating a fault in a system is $\Sigma_f p(f)c(f)$ where $p(f)$ is the probability of each fault and $c(f)$ is the cost of isolating f. $c(f)$ depends on the probing strategy utilized.

Consider the very simple two buffer sequence of Figure 3. Assume A and B are equally likely to fault and $g = .9$. As we

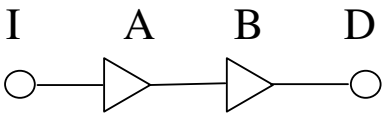


Figure 3: Two buffers in sequence. One is intermittent.

assume the input is given, and that a faulty observation was observed, there is only one measurement point, the output of A that can provide any useful information.

In this simple case there is no choice of measurement point. First, consider the case where B is intermittent. The output of A will always be correct. Following the same line of development as Section 4 after n measurements:

$$p_{n+1}([A]) = \frac{g^n}{g^n + 1},$$

$$p_{n+1}([B]) = \frac{1}{g^n + 1}.$$

To achieve the probability of misdiagnosis to be less than e:

$$\frac{1}{g^n + 1} < e.$$

Solving:

$$n = \frac{\log(1 - e) - \log e}{\log g}.$$

As $e \approx 0$:

$$n = \frac{\log(1 - e)}{\log g}.$$

With $g = .9$ and $e = .01$, $n = 43.7$. Now consider the case where A is faulted. Until the fault is manifest, A will be outputting a good value and the results will look the same as the case where B is faulted. Roughly, the fault will be manifest in,

$$\frac{1}{2 - 2g} = 4.5,$$

samples. Thus, the expected cost of diagnosing Figure 3 is roughly 24.1. Our algorithm obtains a similar result. Again we see that it is considerably more expensive to isolate intermittents further from the input(s).

8 Probing strategy

To successfully isolate the faulty intermittent component, the diagnoser must propose measurements. In conventional model-based diagnosis, (following the myopic strategy) the best probe to make next is the one which minimizes,

$$H = - \sum_{\mathcal{D} \in DIAGNOSES} p(\mathcal{D}) \log p(\mathcal{D}). \quad (4)$$

Even though the definition of $p(c)$ is different than in conventional model-based diagnosis, we still want to probe those variables which maximize the likelihood of distinguishing between the possible intermittent faults. The goal is the same — to lower the posterior probability of failure for all but one of the components. A myopic minimum entropy approach will guide probing to this goal. As we are only considering singlefaults, we seek to minimize:

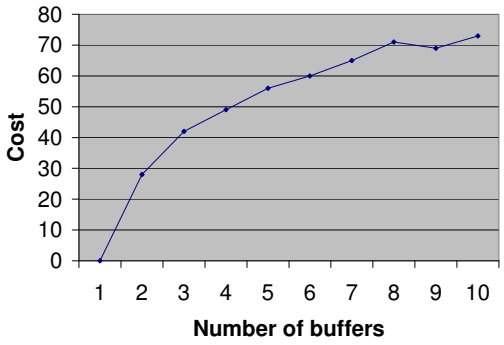


Figure 4: Expected cost of isolating an intermittent buffer in a row of n.

Figure 4 plots expected cost of isolating a single intermittent buffer in a sequence of buffers using the minimum entropy strategy. The graph illustrates that, as is the case in the non-intermittents, groups of components are simultaneously exonerated so that the diagnostic cost grows roughly logarithmic with circuit size. Note that the minimum entropy strategy is much better than the simple approach of Section 4.

Table 2: Result of probing strategy, with $A1$ failing at $g = .9$. The final column is the entropy of the distribution, a measure of how closely the correct diagnosis has been isolated.

i	A1	A2	O1	X1	X2	obs	E
2	0.40	0.20	0.40	0	0	x=0	1.52
3	0.25	0.25	0.50	0	0	y=0	1.50
4	0.27	0.18	0.55	0	0	x=0	1.44
5	0.20	0.20	0.60	0	0	y=0	1.37
6	0.21	0.16	0.63	0	0	x=0	1.31
7	0.17	0.17	0.67	0	0	y=0	1.25
8	0.17	0.14	0.69	0	0	x=0	1.20
9	0.14	0.14	0.71	0	0	y=0	1.15
10	0.15	0.12	0.73	0	0	x=0	1.11
11	1.00	0	0	0	0	y=1	0

If inputs vary, a slightly different approach is needed. It is computationally impractical to find the variable to measure next which provides the maximum expected information gain over all possible system inputs. Instead, we employ a heuristic to ensure measuring the best variable which is directly adjacent to some remaining suspect component. This avoids the possible suboptimal situation where the particular new inputs make the proposed measurement insensitive to any of the faults (e.g., suppose the symptomatic value propagated through a known good and-gate, and if the other input of this and-gate were 1 when the fault was first observed and was 0 in later inputs, measuring the output of that and gate would be useless for those input vectors, and it would take a many samples to isolate the faulty component).

9 Extended example of probing

We simulate the intermittent fault by injecting an incorrect value at times sampled from a lognormal distribution with mean of .9.

Table 2 presents the sequence of probes to isolate a fault in $A1$. $A1$'s fault is manifest at $i = 11$. Notice that failing components closer to the input are easier to isolate as there are fewer confounding effects of intermediate components through which the signals must propagate before reaching a suspect component.

Table 4 presents the sequence of probes to isolate a fault in $O1$. This example illustrates the challenge in isolating faulty components further from the inputs. Although $O1$ can never be identified as failing with probability 1, the table shows that repeated observations drive $O1$'s probability asymptotically to 1.

10 Benchmarks

This section presents results of our algorithm on some standard benchmarks in model-based diagnosis. Our examples are drawn from a widely available combinatorial logic test suite from ISCAS-85 [Brglez and Fujiwara, 1985]. The acid test of performance is how many samples are required to isolate the faulty intermittent component.

Table 3: The first column is the ISCAS-85 circuit name. The second column is the number of distinct gates of the circuit. The remaining two columns are computed by hypothesizing each faulty component with a 1 switched to 0, and a 0 switched to a 1. For each fault, a single (constant) test-vector is found which gives rise to an observable symptom for one of the outputs. For each row, for each component, two simulations are performed. The next column is the average number probes needed to isolate the faulty non-intermittent component. The final column is the average number of probes needed to isolate the intermittent component with misdiagnosis $p < 1 - e$ with $e = .1$. The intermittent fault is manifest by sampling from a lognormal distribution with mean .9.

circuit	components	noni-cost	i cost
c17	6	2.7	36
c432	160	5.5	92
c499	201	5.3	446
c880	384	5.2	114
c1355	546	6.9	554

Table 4: Result of probing strategy, with O1 failing at $g = .9$, (1 in 10 samples).

i	A1	A2	O1	X1	X2	obs	E
2	0.400	0.200	0.400	0	0	x=0	1.52
3	0.250	0.250	0.500	0	0	y=0	1.50
4	0.273	0.182	0.545	0	0	x=0	1.44
5	0.200	0.200	0.600	0	0	y=0	1.37
6	0.211	0.158	0.632	0	0	x=0	1.31
7	0.167	0.167	0.667	0	0	y=0	1.25
8	0.172	0.138	0.690	0	0	x=0	1.20
9	0.143	0.143	0.714	0	0	y=0	1.15
10	0.146	0.122	0.732	0	0	x=0	1.11
20	0.084	0.076	0.840	0	0	x=0	0.80
40	0.046	0.043	0.911	0	0	x=0	0.52
50	0.037	0.036	0.927	0	0	x=0	0.45
100	0.019	0.019	0.962	0	0	x=0	0.27
200	0.010	0.010	0.980	0	0	x=0	0.16
499	0.004	0.004	0.992	0	0	y=0	0.07

Table 3 lists the average cost of diagnosing circuits in the ISCAS-85 benchmark. Not surprisingly, the number of samples needed to isolate intermittent faults scales faster than the cost of identifying non-intermittent faults.

11 Related Work

There has been relatively little work in the model-based diagnosis community on intermittent faults. The approach of this paper exploits the notion of exoneration — ruling out components as failing when observed to be behaving correctly. Previous work along this line is the alibi notion of [Raiman, 1990] and of corroboration in [Brown *et al.*, 1982].

[Koren and Kohavi, 1977] converts intermittent diagnosis tasks of combinational logic to dynamic programming. [Contant *et al.*, 2002] presents an intermittent diagnosis approach applicable to Discrete Event Systems.

12 Conclusion and Future Work

This paper has demonstrated that a relatively direct change to conventional model-based diagnosis algorithms can handle intermittent faults that previously were very hard to diagnose. The underlying inference machinery used by most algorithms needs no change.

13 Acknowledgments

Daniel G. Bobrow, Craig Eldershaw, Prateek Sarkar, Bob Price, and Elisabeth de Kleer provided many useful comments.

References

[Brglez and Fujiwara, 1985] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 695–698, June 1985.

[Brown *et al.*, 1982] J. S. Brown, R. Burton, and J. de Kleer. Pedagogical, natural language, and knowledge engineering issues in SOPHIE I, II, and III. In D. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 227–282. Academic Press, New York, 1982.

[Clark, 1978] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978. Also in: Ginsberg, M. L. (ed) *Readings in Nonmonotonic Reasoning* (Morgan Kaufmann, Los Altos, Calif., 1987).

[Contant *et al.*, 2002] O. Contant, S. Lafortune, and D. Teneketzis. Failure diagnosis of discrete event systems: The case of intermittent faults. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 4006–4011, 2002.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.

[de Kleer *et al.*, 1992] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[de Kleer, 1992] J. de Kleer. A hybrid truth maintenance system. PARC Technical Report, January 1992.

[de Kleer, 2006] J. de Kleer. Getting the probabilities right for measurement selection. In *17th International Workshop on Principles of Diagnosis*, pages 141–146, Burgos, Spain, 2006.

[Fromherz *et al.*, 2003] M.P.J. Fromherz, D.G. Bobrow, and J. de Kleer. Model-based computing for design and control of reconfigurable systems. *The AI Magazine*, 24(4):120–130, 2003.

[Koren and Kohavi, 1977] Israel Koren and Zvi Kohavi. Diagnosis of intermittent faults in combinational networks. *IEEE Trans. Computers*, 26(11):1154–1158, 1977.

[Raiman, 1990] O. Raiman. A circumscribed diagnosis engine. In G. Gottlob and W. Nejdl, editors, *Proc. Int. Workshop on Expert Systems in Engineering*, pages 90–101. Springer Verlag LNCS 462, 1990.

Troubleshooting temporal behavior in "combinational" circuits

Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
dekleer@parc.com

Abstract

This paper addresses the challenge of reasoning and diagnosing digital circuits which contain either intentional or unintentional cycles in the combinational logic. Such cycles can lead to oscillatory behavior or convert what seems at first to be a combinational circuit into a sequential one. Such circuits often produce instant contradictions when analyzed at the logical gate level. This paper presents a temporal extension to the GDE framework which makes it possible to analyze and successfully troubleshoot such circuits.

1 Introduction

This paper addresses the challenge of reasoning and diagnosing digital circuits which contain either intentional or unintentional cycles in the combinational logic. Such cycles can lead to oscillatory behavior or convert what seems at first to be a combinational circuit into to a sequential one. Such circuits often produce instant contradictions when analyzed at the logical gate level. This paper presents a temporal extension to the GDE framework [de Kleer and Williams, 1987; de Kleer *et al.*, 1992] to represent signals over time. With these new models its possible to analyze and successfully troubleshoot such circuits.

The temporal analyses presented in this paper extend both the basic GDE-style models and the connection models in [de Kleer, 2007]. In this paper we make the following simplifying assumptions.

- All gate delays are equal.
- Signals take no time to propagate through connections.
- Does not take advantage of non-intermittency axioms [Raiman *et al.*, 1991].

2 Motivating examples

Consider the 3 inverter circuit of Figure 1. It is easy to wire three inverters together — the system is physically realizable and connecting the inverters in this way does not damage them. However, using the usual model for gates described in [de Kleer and Williams, 1987] or the extended models for connections decribed in [de Kleer, 2007], GDE immediately detects a contradiction concluding that at least one of the components or nodes is necessarily faulted. In fact, none of the inverters are faulted. This is a well-known common circuit used to generate clock signals in digital circuits [Wikipedia, 2007].

Table 1: Outputs of the inverters of a ring oscillator after t gate delays. The oscillator takes 6 gate delays to return to its initial state, thus the output is a square wave with a period of 6 times the gate delay.

t	0	1	2	3	4	5	6
A	1	1	1	0	0	0	1
B	1	0	0	0	1	1	1
C	0	0	1	1	1	0	0

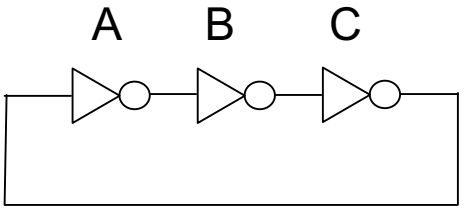


Figure 1: Simple ring oscillator.

Figure 2 illustrates an SR flip-flop. Table 2 describes the truth table for the circuit. The final line in the truth table represents two possible states the flip-flop could be in, and the values of Q and $\overline{Q}$ depend on previous history. If the preceeding inputs were $\overline{S} = 0$ and $\overline{R} = 1$, then Q will become 1 and $\overline{Q}$ will become 0 (corresponding to setting the flip-flop). Symmetrically, if the inputs are flipped, the outputs will be flipped (corresponding to resetting the flip-flop). If the inputs change to 1's then Q and $\overline{Q}$ will retain their previous values. The SR flip-flop is the central way static memory elements are created in VLSI design.

Figure 3 shows a connection short which can cause unintended oscillation in c17 from [Brglez and Fujiwara, 1985]. With the inputs as indicated, the output at O1 should be 0. If O1 is shorted to I2, the circuit will oscillate. Modeling con-

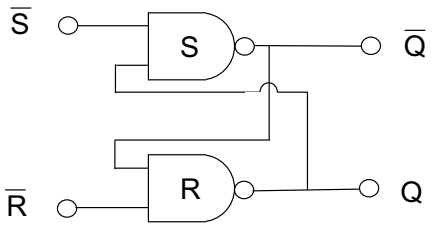


Figure 2: SR flip-flop. It consists of two cross-coupled NAND gates.

Table 2: Truth table for SR flip-flop.

$\overline{R}$	$\overline{S}$	Q	$\overline{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	?	?

nections as in [de Kleer, 2007], O1 will pull down I2 to 0, the output of the nand gate G1 will change from 0 to 1, and thus the output of the nand gate G5 will switch to 1. O1 is shorted to I2, so it will now return to 1. The circuit will continue oscillating in this manner forever. When modeled with GDE and the connection models of [de Kleer, 2007], this circuit with these inputs are completely contradictory and the correct fault is therefore completely eliminated from consideration. Many sets of design rules used in VLSI design try to minimize such hard to isolate faults by not allowing inputs to run adjacent to outputs.

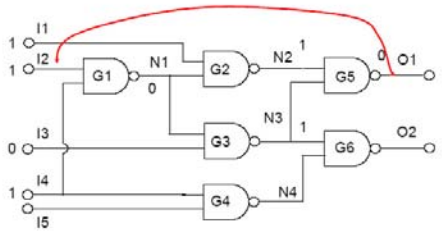


Figure 3: Short circuit causes undesired oscillation.

Figure 4 represents a common failure in CMOS which creates an SR flip-flop unintentionally. This is hard to troubleshoot because a previously simple combinational circuit now has state.

3 Modeling Components

The conventional GDE model for an inverter is:

$$INVERTER(x) \rightarrow \Big[\neg AB(x) \rightarrow [in(x,t) = 0 \equiv out(x,t) = 1]\Big].$$

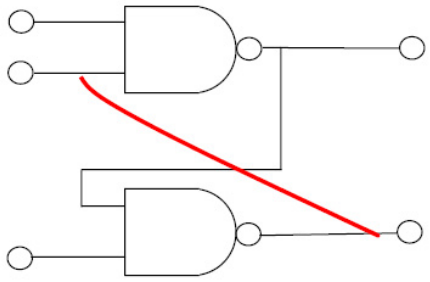


Figure 4: Short circuit which creates memory.

Modeling an inverter as having a delay, the model changes to:

$$INVERTER(x) \rightarrow \Big[\neg AB(x) \rightarrow [in(x,t) = 0 \equiv out(x,t+\Delta) = 1]\Big].$$

To accomodate connection failures, signals on wires must be modeled at a more detailed level [de Kleer, 2007]. Each terminal of a component is modeled with two variables, one which models how the component is attempting to influence its output (roughly analogous to current), and the other which characterizes the result (roughly analogous to voltage). For a correctly functioning node, these voltage-like variables are equal. There are 5, mutually inconsistent, qualitative values for the influence of a component on a node (we refer to these as "drivers").

- $d(-\infty)$ indicates a direct short to ground.
- $d(0)$ pull towards ground (i.e., 0).
- $d(R)$ presents a high (i.e., draws little current) passive resistive load.
- $d(1)$ pull towards power (i.e., 1).
- $d(+\infty)$ indicates a direct short to power.

Intuitively, these 5 qualitative values describe the range of possible current sinking/sourcing behaviors of a component terminal. A direct short to ground can draw a large current inflow. A direct power to ground can drive a large current outflow.

There are three possible qualitative values for the result variable:

- $s(0)$ the result is close enough to ground to be sensed as a digital 0.
- $s(x)$ the result is neither a 0 or 1.
- $s(1)$ the result is close enough to power to be sensed as a digital 1.

With these connection models, the inverter is modeled as:

$$INVERTER(x) \rightarrow \Big[\neg AB(x) \rightarrow$$

$$[s(in(x,t)) = s(0) \rightarrow d(out(x,t)) = d(1)$$
$$\wedge s(in(x,t)) = s(1) \rightarrow d(out(x,t)) = d(0)$$
$$\wedge d(in(x,t)) = d(R)$$
$$\wedge d(out(x,t)) = d(0) \vee d(out(x,t)) = d(1)]\Big].$$

Modeling the inverter to more accurately describe both temporal and causal behavior:

$$INVERTER(x) \rightarrow$$
$$\Big[\neg AB(x) \rightarrow$$
$$[s(in(x,t)) = s(0) \rightarrow d(out(x,t+\Delta)) = d(1)$$
$$\wedge s(in(x,t)) = s(1) \rightarrow d(out(x,t+\Delta)) = d(0)$$
$$\wedge d(in(x,t)) = d(R)$$
$$\wedge d(out(x,t+\Delta)) = d(0) \vee d(out(x,t+\Delta)) = d(1)]\Big].$$

4 Representing Time

Our implementation is based on the ATMS/HTMS structure. Each time instant is modeled by an explicit assumption $t = i$ and any two time assumptions contradict each other. These assumptions separate all inferences into their respective times. However, to instantiate the component models which incorporate evolving time, a non-monotonic ATMS rule is required. An instantiation of the model for a particular inverter A is encoded in the following two clauses (the implication $x \wedge y \wedge ... \rightarrow z$ is equivalent to the logical clause $\neg x \vee \neg y \vee ... \rightarrow z$ and the literal $\neg(q = 0)$ is equivalent to literal $q = 1$):

$$AB(A) \vee in(A,t) = 1 \vee [out(A,t+\Delta) = 1]$$
$$AB(A) \vee in(A,t) = 0 \vee [out(A,t+\Delta) = 0].$$

However, time is implicitly represented, by explict assumptions. So the assumption for t_i would be implicit in any deduction for $in(A,t)$. Any deduction of $in(A)$ will have a single t_i assumption. We introduce a new modal operator N which specifies that its argument holds in the next point beyond its antecedent. So the two inverter clauses become:

$$AB(A) \vee in(A) = 1 \vee N[out(A) = 1]$$
$$AB(A) \vee in(A) = 0 \vee N[out(A) = 0].$$

These clauses are modeled in the ATMS by the following non-monotonic inference rule: Every (final or intermediate) prime implicate of the form,

$$AB(c_1) \vee \cdots \vee AB(c_n) \vee \neg(t = i) \vee N(x),$$

is rewritten:

$$AB(c_1) \vee \cdots \vee AB(c_n) \vee \neg(t = i+1) \vee x.$$

This rule is non-monotonic because the result no longer depends on the existance of the previous time instant. Without this inference rule, it would not be possible to model evolution of time as time is inherently non-monotonic.

An advantage of this scheme to represent time is that it is not necessary to make multiple copies of component models for each time. Once the clauses representing the system model are represented, all further propagations are accomplished through the ATMS.

Suppose the input to the first inverter of the ring oscillator is observed to be 0 at t_1. Inference proceeds as follows:

$$AB(A) \vee \neg(t = 1) \vee N[out(A) = 1]$$
$$AB(A) \vee \neg(t = 2) \vee [out(A) = 1]$$
$$AB(A) \vee AB(B) \vee \neg(t = 2) \vee N[out(B) = 0]$$
$$AB(A) \vee AB(B) \vee \neg(t = 3) \vee [out(B) = 0]$$
$$AB(A) \vee AB(B) \vee AB(C) \vee \neg(t = 3) \vee N[out(C) = 1]$$
$$AB(A) \vee AB(B) \vee AB(C) \vee \neg(t = 4) \vee [out(C) = 1]$$
$$AB(A) \vee AB(B) \vee AB(C) \vee \neg(t = 4) \vee N[out(A) = 0]$$
$$AB(A) \vee AB(B) \vee AB(C) \vee \neg(t = 5) \vee out(A) = 0$$

All subsequent inferences follow the same pattern. No new assumptions will be added.

5 Modeling the ring oscillator

Applying the temporal models produces the envisionment of Figure 5. The usual states associated with a ring oscillator are described by the states on the outside of the envisionment figure. This corresponds to a stable oscillation of period 6 gate delays. This oscillation is stable because there are no ambiguous transitions. However, there are two spurious states in a meta-stable oscillation at the center of the envisionment diagram. Over time one of the gate delays will be slightly longer or shorter and the others and the system will transition to one of the 6 stable states. All transitions out of 000 and 111 are ambiguous because multiple transitions are happening simultaneously. While in the 000-111 oscillation will look like Figure 10. For this reason, most practical ring oscillators have an enable input (Figure 8 so that the circuit starts in a known stable state.

6 Signals

The notation $f(g,t)$ (e.g., $in(A,t)$) to represent values is clumsy approach to represent a changing value for time (a fluent). We introduce a notion of signal:

Definition 1 *A signal represents the evolution of a value over time (at the granularity of a gate delay). It is represented by a sequence of symbols drawn from 0,1,?,>,*. 0,1 indicate their respective values, "?" indicates the value is unknown, ">" indicates a large (unknown) number of gate delays, and "*" means that the following pattern repeats. A signal may have only one "*" and it must occur at the earliest possible place. We call signals containing a * followed by both 1's and 0's as definitely oscillatory. We call signals in which * is followed by exactly one symbol 0 or 1 as steady.*

Examples of valid signals are:

- *0 the steady signal always 0.
- >*1 the steady signal eventually 1.

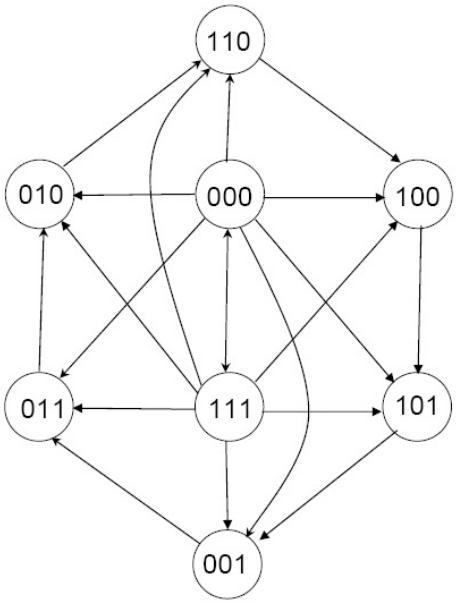


Figure 5: Envisionment for the ring oscillator. The vertices are labeled with the values for the outputs of A, B, C respectively.

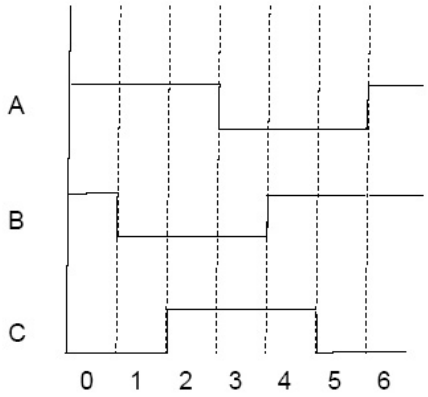


Figure 7: Desired output of the ring oscillator.

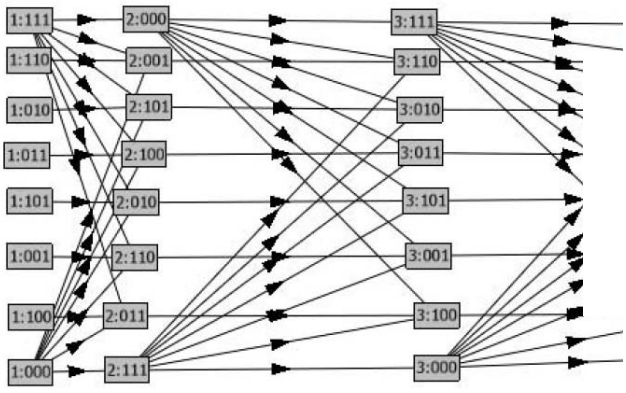


Figure 6: Qualitative simulation of the ring oscillator. The vertices are labeled: t:A,B,C where t is the time in gate delays and A,B,C are the outputs of the respective inverters.

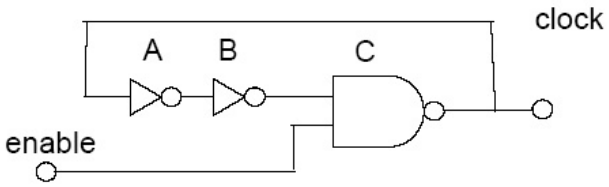


Figure 8: A better designed ring oscillator. When the clock is enabled, the ring oscillator will start in a known good state: 011

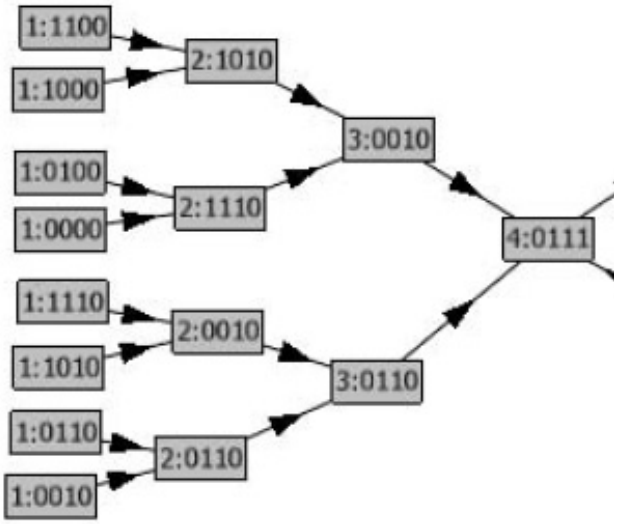


Figure 9: The oscillatory behavior of the better ring oscillator. Vertices are labeled t:nnnn: t is the time, nnnn are the outputs of gates A,B,C and the enable input. The enable signal is 000*1.

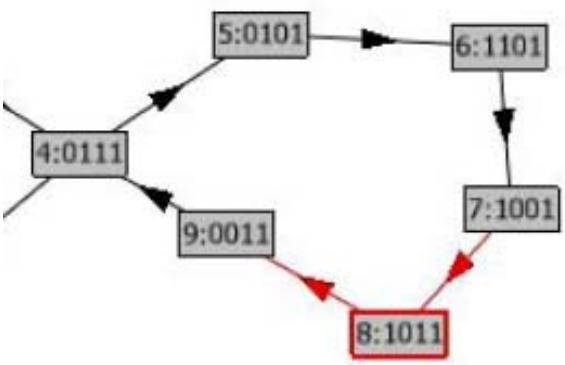


Figure 11: Qualitative simulation of the better ring oscillator. Vertices are labeled t:nnnn: t is the time, nnnn are the outputs of gates A,B,C and the enable input. The enable signal is 000*1.

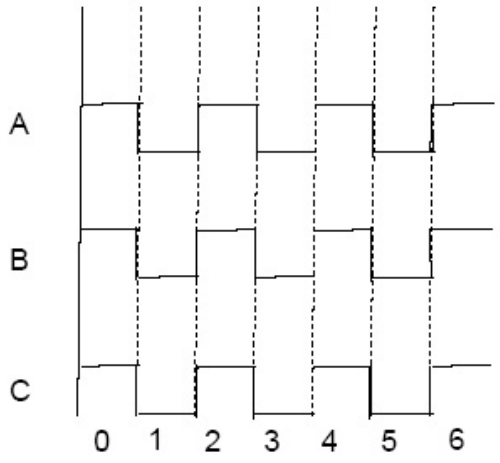


Figure 10: Undesired spurious oscillation of the ring oscillator.

- >*000111 the oscillatory output of a 3 gate ring oscillator

Signals are used to describe all of a system's variables. Figure 11 shows a continuation of the simulation of Figure 9. They show that the resultant signal at the output of the ring oscillator is: ?*111000. Our extended GDE, infers not just the values of variables at particular time points (such as $in(A, t_1)$) but also all the most general signals (presuming all signals eventually have a repeating pattern). The assumptions supporting this derivation are the assumptions of its constituent components:

$$AB(A) \vee AB(B) \vee AB(C) \vee (C =? * 111000).$$

On the surface, these derivations don't seem to have much diagnostic power, as in most cases the signals will be depend on the non-ABnormal behavior of all the components of the system.

To illustrate the diagnostic power of inferring complex signals consider a slightly more complex ring oscillator consisting of 4 inverters and one nand gate as illustrated in Figure 12. Assume the input is 00000*1. GDE will compute an output of >*0000011111. Suppose the output is observed to be *0. Then we immediately have the conflict:

$$AB(A) \vee AB(B) \vee AB(C) \vee AB(D) \vee AB(E) \vee AB(G).$$

The obsevation *0 is propagated (as it would if a simple 0 were observed). If all components are equally likely to fail, the usual GDE probing strategy will choose the next measurement (either the output of B or C) and gate causing the failure to oscillate can be isolated as usual. More intuitively,

as we are observing the system when all observations and inputs are steady, the circuit looks like that of Figure 13 which can be easily diagnosed.

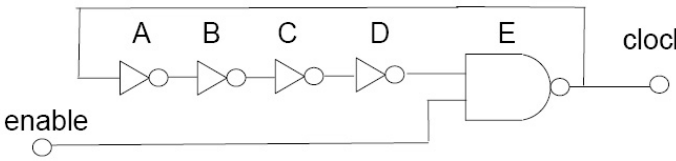


Figure 12: A 5 gate ring oscillator.

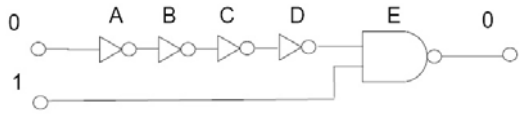


Figure 13: A faulty 5 gate ring oscillator observed at quiescent.

Consider the circuit of Figure 14 which combines an SR flip-flop with a ring oscillator. The purpose of this circuit is to ensure the clock is on (it might be on already) when the flip-flop sees a single negative pulse at S. Suppose the clock output is observed to be quiescent 0. Observed at quiescent every component is obeying its io-behavior: S=*0, R=*1, C=*0, B=*1, A=*1, yet the circuit is faulted. With the two given inputs at quiescence being 1, the outputs of S and R cannot be definitively infered. Even though we observe S to be *0, at quiescence that cannot be inferred. So that observation provides no information about whether S or R are faulted or not. Fortunately, the fault at S and R can be directly determined by the basic propagation of signals. Given the observations as shown, the signal at the output of S should be ???1* hence one of S or R are faulted. For example, S stuck at 0, or R stuck at 1 would explain the observation. The reason its hard to diagnose at quiescence is that S stuck at 0 is the same output as it were working correctly. The fault occurred in the past.

7 Related work

[Dague *et al.*, 1991] focused on troubleshooting analog oscillators which stopped oscillating. [Kaul *et al.*, 1994] focused on simulating CMOS designs with QSIM [Kuipers, 1986]. Neither approach generalized to explicitly recording assumptions of good behavior in order to isolate the malfunctioning component through more observations. More details of digital circuit behavior can be found in [Weste and Eshraghian, 1993] and [Johnson and Graham, 1993].

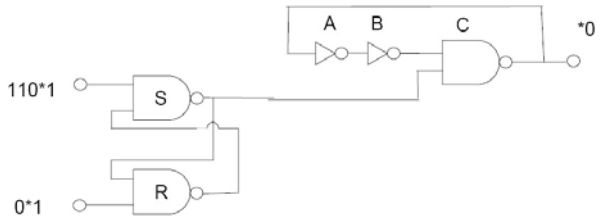


Figure 14: A flip-flop combined with a ring oscillator. The figure shows the initial inputs and it is observed at quiesence: S=*0, R=*1, C=*0, B=*1, A=*1 which is consistent with all the component models, yet the output is incorrect.

8 Conclusions

This research has extended the GDE framework with a simple vocabulary to represent signals over time. The same basic architecture and inferential machinery as GDE can be used to propagate these generalized signals. Discrepencies between observed and predicted generalized signals can guide diagnosis to isolate the faulty system component(s). These generalized signals capture temporal behavior over time so they can be used to troubleshoot sequential circuits (e.g., containing flip-flops) as well. Sequential circuits can be hard to diagnose because no symptom maybe observable at quiescence. With these extensions GDE can be used to troubleshoot a far wider range of circuits than previously.

References

[Brglez and Fujiwara, 1985] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 695–698, June 1985.

[Dague *et al.*, 1991] P. Dague, O. Jehl, P. Deves, P. Luciani, and P. Taillibert. When oscillators stop oscillating. In *Proc. 12th Int. Joint Conf. on Artificial Intelligence*, pages 1109–1115, Sydney, Australia, August 1991.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.

[de Kleer *et al.*, 1992] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[de Kleer, 2007] J. de Kleer. Modeling when connections are the problem. In *Proc 20th IJCAI*, pages 311–317, Hyderabad, India, 2007.

[Johnson and Graham, 1993] Howard W. Johnson and Martin Graham. *High-speed digital design: a handbook of black magic*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[Kaul *et al.*, 1994] Neeraj Kaul, Gautam Biswas, and Bharat Bhuva. An artificial intelligence approach to multi-level

mixed-mode qualitative simulation of cmos ics. *Comput. Electr. Eng.*, 20(5):369–382, 1994.

[Kuipers, 1986] B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, September 1986. Also in: D. S. Weld and J. de Kleer (eds) *Readings in Qualitative Reasoning about Physical Systems* (Morgan Kaufmann, Los Altos, Calif., 1990).

[Raiman *et al.*, 1991] O. Raiman, J. de Kleer, V. Saraswat, and M. H. Shirley. Characterizing non-intermittent faults. In *Proc. 9th National Conf. on Artificial Intelligence*, pages 849–854, Anaheim, CA, July 1991.

[Weste and Eshraghian, 1993] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI design: a systems perspective, second edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[Wikipedia, 2007] Wikipedia. Ring oscillator — Wikipedia, the free encyclopedia, 2007. [Online; accessed 12-February-2007].

Coverage Techniques for Checking Temporal-Observation Subsumption

Andrea Ducoli and **Gianfranco Lamperti** and **Emanuele Piantoni** and **Marina Zanella**

Dipartimento di Elettronica per l'Automazione, Università di Brescia
Via Branze 38, 25123 Brescia, Italy
Tel: +39 030 37 15 491, Fax: +39 030 38 00 14
{aducoli.52602, epiantoni.52704}@studenti.unibs.it, {lamperti, zanella}@ing.unibs.it

Abstract

Similarity-based diagnosis of large discrete-event systems is supported by reuse of knowledge generated for solving previous diagnostic problems. Such knowledge is cumulatively stored in a knowledge base, when the diagnostic session is over. When a new diagnostic problem is to be faced, the knowledge base is queried in order to possibly find a similar, reusable problem. Checking problem-similarity requires, among other constraints, that the observation relevant to the new problem subsume the observation relevant to the problem in the knowledge base. However, checking observation-subsumption, following its formal definition, is time and space consuming. The bottleneck lies in the generation of a nondeterministic automaton, its subsequent transformation into a deterministic one (the index space of the observation), and a regular-language containment-checking. In order to speed up the diagnostic process, an alternative technique is proposed, based on the notion of coverage. Besides being effective, subsumption-checking via coverage is also efficient because no index-space generation or comparison is required. Evidence from experimental results supports this claim.

Introduction

Discrete-event systems (DESs) (Cassandras & Lafortune 1999) are dynamic systems, typically modeled as networks of reactive components. Each component is a communicating automaton (Brand & Zafiropulo 1983) that reacts to input events by state-transitions which possibly generate new events towards other components. Diagnosis of DESs is a challenging task that has been tackling since a decade via different approaches, either based on artificial intelligence (Pencolé 2000; Rozé & Cordier 2002; Console, Picardi, & Ribaudo 2002; Pencolé & Cordier 2005) or automatic control techniques (Sampath *et al.* 1995; 1996; Chen & Provan 1997; Sampath, Lafortune, & Teneketzis 1998; Zad, Kwong, & Wonham 1999; Cassandras & Lafortune 1999; Lunze 2000; Debouk, Lafortune, & Teneketzis 2000; Schullerus & Krebs 2001). Within the domain of diagnosis of active systems, a class of asynchronous DESs (Baroni *et al.* 1999; Lamperti & Zanella 2003), an approach has been proposed that is based on similarity techniques (Lamperti & Zanella 2006; Cerutti *et al.* 2007) with the aim of pursuing reuse of knowledge when solving a diagnostic problem. The idea is to store into a knowledge-base the data structures generated for solving each diagnostic problem. When a new problem is to be faced, instead of solving it from scratch, the knowledge base is first browsed in order to find a previously-solved diagnostic problem that is 'compatible' with the new one. If so, the knowledge relevant to the old problem can be exploited to solve the new problem, thereby speeding up the diagnostic process. Among other constraints, such compatibility requires that the observation relevant to the problem in the knowledge base subsume the observation relevant to the new problem. Such an observation is temporal in nature, and is represented by a DAG. The problem lies on the mode in which subsumption is checked, which, according to the definition of subsumption, is based on a containment relationship between the regular languages of the *index spaces* of the observations. The index space is a deterministic automaton whose generation (and comparison) may require considerable computational resources. So, an alternative approach for checking observation-subsumption is required, possibly avoiding index-space manipulation, by reasoning on the specific properties of the observations.

Background

When a DES reacts, it generates a sequence of observable labels, called the *signature* of the reaction. However, what is actually perceived by the external observer is a *relaxation* of the signature $\mathbb{S}$. Such a relaxation is called a *temporal observation*. Formally, let $\mathfrak{L}$ be a finite domain of labels, possibly including the *null* label ϵ. A temporal observation is a (not necessarily connected) DAG

$$\mathcal{O} = (\mathcal{N}, \mathfrak{L}, \mathcal{A}) \tag{1}$$

where $\mathcal{N}$ is the set of nodes, with each $N \in \mathcal{N}$ being marked with a non-empty subset of $\mathfrak{L}$, and $\mathcal{A} : \mathcal{N} \mapsto 2^{\mathcal{N}}$ is the set of arcs. A '$\prec$' temporal precedence relationship among nodes of the graph is defined as follows:

- If $N \mapsto N' \in \mathcal{A}$ then $N \prec N'$;
- If $N \prec N'$ and $N' \prec N''$ then $N \prec N''$;
- If $N \mapsto N' \in \mathcal{A}$ then $\nexists N'' \in \mathcal{N}$ $(N \prec N'' \prec N')$.

The set of labels marking a node N is the *extension* of N, written $\|N\|$. Thus, the relaxation of the signature $\mathbb{S}$ into $\mathcal{O}$ involves two kinds of uncertainty:

- *Logical uncertainty*: each observable label in the signature $\mathbb{S}$ is perceived as a set of candidate labels, possibly including the null label ϵ. Thus, all labels in $\|N\|$ but one are *spurious*, with just one being the *actual label*.[1]
- *Temporal uncertainty*: the absolute temporal ordering of the signature $\mathbb{S}$ is relaxed to partial temporal ordering, such that if $N \prec N'$ in $\mathcal{O}$, where ℓ and ℓ' are the actual labels in N and N', respectively, then ℓ precedes ℓ' in $\mathbb{S}$.

As such, $\mathcal{O}$ implicitly incorporates several *candidate signatures*, where each candidate is determined by selecting one label from each node in $\mathcal{N}$ without violating the temporal constraints imposed by the precedence relationships. Among such candidates is the actual signature $\mathbb{S}$. Like for nodes, all candidate signatures but one are spurious. The mode in which the signature $\mathbb{S}$ demeans to observation $\mathcal{O}$ is assumed to be unknown.[2] As explained in (Lamperti & Zanella 2002), such a degradation may be caused by the multiplicity of the communication channels that convey observable labels from the system to the observer (temporal uncertainty), and to noise (logical uncertainty). However, although unknown, $\mathbb{S}$ is assumed to be preserved within $\mathcal{O}$.

Example 1. Shown in Fig. 1 are the graphs of two temporal observations, namely, from left to right, $\mathcal{O}_1 = (\mathcal{N}_1, \mathcal{L}_1, \mathcal{A}_1)$ and $\mathcal{O}_2 = (\mathcal{N}_2, \mathcal{L}_2, \mathcal{A}_2)$, where $\mathcal{N}_1 = \{N_1, \ldots, N_5\}$, $\mathcal{N}_2 = \{N'_1, \ldots, N'_4\}$, $\mathcal{L}_1 = \{a, b, c, d, f, \epsilon\}$, and $\mathcal{L}_2 = \{a, b, c, d, \epsilon\}$. For instance, $\mathcal{O}_2$, which includes four nodes, exhibits both logical and temporal uncertainty. Accordingly, N'_1 incorporates the first observable label, namely a. Then, either N'_2 or N'_3 follows, each of which involves two candidate labels, where ϵ is null. The last generated node is N'_4, with a and ϵ being the final candidate labels. Thus, based on the logical content of each node and on the partial temporal relationships among nodes, it is easy to show that the extension of the observation, namely $\|\mathcal{O}_2\|$, includes the candidate signatures ac, ad, abc, abd, aca, ada, acb, adb, $abca$, $abda$, $acba$, $adba$, each of which is obtained by selecting one label for each node without violating the temporal constraints, where the null label ϵ is removed.

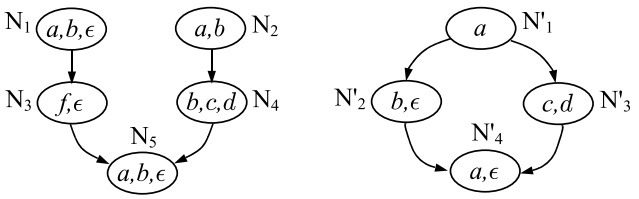


Figure 1: Observations $\mathcal{O}_1$ (left) and $\mathcal{O}_2$ (right).

Within the diagnostic process it is inconvenient to reason on the observation $\mathcal{O}$ as is, mostly because the explanation-oriented diagnostic reasoning requires some sort of observation-indexing. Such an indexing is more naturally performed based on a surrogate of the observation, called the *index space*, namely $Isp(\mathcal{O})$. This is a deterministic automaton with the property that its regular language is the extension of $\mathcal{O}$,

$$Lang(Isp(\mathcal{O})) = \|\mathcal{O}\|. \quad (2)$$

In other words, the set of strings generated by each path in $Isp(\mathcal{O})$, from the initial state to a final state, equals the set of candidate signatures relevant to $\mathcal{O}$. As detailed in (Cerutti *et al.* 2007), the generation of the index space of $\mathcal{O}$ requires two steps, namely:

- Yielding the nondeterministic automaton, called the *prefix space* of $\mathcal{O}$, where each node identifies the set of consumed nodes in $\mathcal{N}$ up to now;
- Generating the deterministic automaton equivalent to the prefix space, in fact the index space.[3]

Furthermore, as explained shortly, the role of the index space comes into view for checking observation-subsumption too.

Example 2. Shown in Fig. 2 are the index spaces of observations $\mathcal{O}_1$ (left) and $\mathcal{O}_2$ (right) displayed in Fig. 1. It is easy to check that the regular language of each index space equals the extension of the relevant observation (the set of candidate signatures), where each string of the language corresponds to a path in the index space, from the initial state to one of the final states (with the latter being double circled in the figure). In particular, Example 1 offers evidence that $Lang(Isp(\mathcal{O}_2)) = \|\mathcal{O}_2\|$.

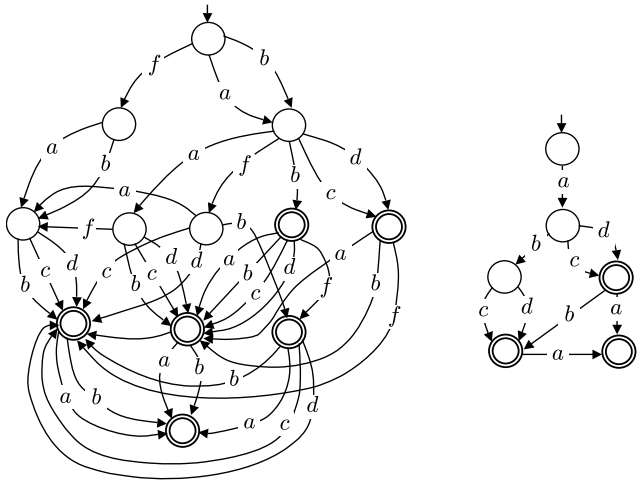


Figure 2: Index spaces $Isp(\mathcal{O}_1)$ (left) and $Isp(\mathcal{O}_2)$ (right).

In similarity-based diagnosis of DESs (Lamperti & Zanella 2006), it is essential to understand if the solution of the diagnostic problem $\wp'$ at hand can be supported by the knowledge yielded for solving a previous (different) diagnostic problem $\wp$, with the latter being stored in a knowl-

[1] If the actual label is ϵ, it means that no label was actually generated by the system. Note how the extension of a node cannot be the singleton $\{\epsilon\}$.

[2] Otherwise, in principle, we might distill $\mathbb{S}$ from $\mathcal{O}$, thereby disregarding $\mathcal{O}$ in the diagnostic process.

[3] Being equivalent, both the prefix space and the index space share the same regular language (Hopcroft, Motwani, & Ullman 2006).

edge base. Among other constraints, reuse of $\wp$ can be exploited only if the observations $\mathcal{O}'$ and $\mathcal{O}$ relevant to $\wp'$ and $\wp$, respectively, are linked by a subsumption relationship,

$$\mathcal{O} \Supseteq \mathcal{O}' \tag{3}$$

namely, only if $\mathcal{O}$ *subsumes* $\mathcal{O}'$. The subsumption relationship is defined in terms of regular-language containment, relevant to the corresponding index spaces, precisely:

$$Lang(Isp(\mathcal{O})) \supseteq Lang(Isp(\mathcal{O}')). \tag{4}$$

This means that $\mathcal{O}$ subsumes $\mathcal{O}'$ iff the set of candidate signatures of $\mathcal{O}$ includes all the candidate signatures of $\mathcal{O}'$.

The reason why observation subsumption supports reuse can be roughly explained as follows. The solution of $\wp$ yields an automaton μ, a sort of diagnoser, where each state is marked by a set of diagnoses and each transition is marked by a label in $\mathcal{L}$. The language of μ is the subset of the signatures relevant to $\mathcal{O}$ that comply with the model of the system, namely, $Lang(\mu) \subseteq \|\mathcal{O}\|$. The same applies to a new problem $\wp'$ relevant to $\mathcal{O}'$. However, if $\mathcal{O} \Supseteq \mathcal{O}'$, that is, $\|\mathcal{O}\| \supseteq \|\mathcal{O}'\|$, then $Lang(\mu) \supseteq Lang(\mu')$. In other words, μ contains all the signatures of μ'. This allows the diagnostic engine to reuse μ in order to generate μ' based on $\mathcal{O}'$. The advantage stems from the fact that such an operation is far more efficient than generating μ' from scratch, which would require heavy model-based reasoning.

Example 3. With reference to observations $\mathcal{O}_1$ and $\mathcal{O}_2$ outlined in Fig. 1, and the relevant index spaces in Fig. 2, it is easy to check that $\mathcal{O}_1 \Supseteq \mathcal{O}_2$, that is, $\|\mathcal{O}_1\| \supseteq \|\mathcal{O}_2\|$. In other words, each string in $Lang(Isp(\mathcal{O}_2))$ is also a string in $Lang(Isp(\mathcal{O}_1))$.

The problem with observation-subsumption checking lies on the fact that, establishing whether we can exploit the knowledge for solving $\wp$, in order to solve $\wp'$, requires a considerable amount of computational resources. Specifically, we need first generate $Isp(\mathcal{O}')$ and, subsequently, compare $Lang(Isp(\mathcal{O}'))$ with each $Lang(Isp(\mathcal{O}))$ in the knowledge base, in the hope of finding a subsuming observation $\mathcal{O}$. Such an approach, based on the generation of the index space and on regular-language containment-checking, may be prohibitive in real applications. In order to cope with this complexity, we need some alternative checking-techniques.

Checking Subsumption

The systematic nature of checking based on the formal definition of subsumption stems primarily on its lack of prospection (short-sightedness). As a matter of fact, such a *systematic* technique does not perform any kind of reasoning on the given observations. Assume the problem of testing $\mathcal{O} \Supseteq \mathcal{O}'$, namely the *checking problem*. The idea is to find out some conditions that either imply or are implied by such a relationship. If these conditions can be checked using a reasonable amount of computational (space and time) resources, then chances are that we can give an answer to the checking problem efficiently. Specifically, if a necessary condition N_c is violated, then the answer to the checking problem will be *no*. Dually, if a sufficient condition S_c holds, then the answer will be *yes*. However, if either N_c holds or S_c is violated, then the checking problem remains unanswered. Necessary conditions and sufficient conditions relevant to the checking problem are given in Theorem 1 and Theorem 2, respectively. As shown shortly, these conditions are eventually incorporated within Algorithm 1 (see below).

Theorem 1. *Let $\mathcal{O} = (\mathcal{N}, \mathcal{L}, \mathcal{A})$ and $\mathcal{O}' = (\mathcal{N}', \mathcal{L}', \mathcal{A}')$ be two temporal observations. Let n and n' be the number of nodes in $\mathcal{N}$ and $\mathcal{N}'$, respectively. Let n_ϵ and n'_ϵ be the number of nodes that include the null label ϵ in $\mathcal{N}$ and $\mathcal{N}'$, respectively. Let $\mathcal{M}$ and $\mathcal{M}'$ be the multisets of observable labels occurring in $\mathcal{O}$ and $\mathcal{O}'$, respectively. Then, $\mathcal{O}$ subsumes $\mathcal{O}'$ only if the following conditions hold:*

$$n \geq n' \tag{5}$$

$$n_\epsilon - n'_\epsilon \geq n - n' \tag{6}$$

$$\mathcal{M} \supseteq \mathcal{M}'. \tag{7}$$

Corollary 1.1. *$\mathcal{O}$ subsumes $\mathcal{O}'$ only if*

$$\mathcal{L} \supseteq \mathcal{L}'. \tag{8}$$

Example 4. In Example 3 we have shown that $\mathcal{O}_1$ subsumes $\mathcal{O}_2$, where such observations are displayed in Fig. 1. Hence, the conditions relevant to Theorem 1 are expected to hold for $\mathcal{O}_1$ and $\mathcal{O}_2$. We have $n_1 = 5, n_2 = 4, n_{1\epsilon} = 3, n_{2\epsilon} = 2$. As a matter of fact, both conditions (5) and (6) hold. Moreover, since $\mathcal{M}_1 = [a, a, a, b, b, b, b, c, d, f, \epsilon, \epsilon, \epsilon]$ and $\mathcal{M}_2 = [a, a, b, c, d, \epsilon, \epsilon]$, condition (7) holds too.

The conditions necessary for subsumption stated in Theorem 1 can be easily checked. Thus, they correspond to the first actions of the checking algorithm. If one of them is violated, the check terminates immediately with a negative answer. Otherwise, the check continues by testing a sufficient condition of subsumption based on the notion of coverage given in Definition 1. Roughly, $\mathcal{O}$ covers $\mathcal{O}'$ when $\mathcal{O}$ is a relaxation of $\mathcal{O}'$, inasmuch as an observation is a relaxation of a system signature.

Definition 1. (Coverage) *Let $\mathcal{O} = (\mathcal{N}, \mathcal{L}, \mathcal{A})$ and $\mathcal{O}' = (\mathcal{N}', \mathcal{L}', \mathcal{A}')$ be two temporal observations, where $\mathcal{N} = \{N_1, \ldots, N_n\}$ and $\mathcal{N}' = \{N'_1, \ldots, N'_{n'}\}$. We say that $\mathcal{O}$* covers *$\mathcal{O}'$, written $\mathcal{O} \trianglerighteq \mathcal{O}'$, iff there exists a subset $\bar{\mathcal{N}}$ of $\mathcal{N}$, with $\bar{\mathcal{N}} = \{\bar{N}_1, \ldots, \bar{N}_{n'}\}$ having the same cardinality as $\mathcal{N}'$, such that, denoting $\mathcal{N}^\epsilon = \left(\mathcal{N} - \bar{\mathcal{N}}\right)$, we have:*

(1) (ϵ-coverage): $\forall N \in \mathcal{N}^\epsilon$ $(\epsilon \in \|N\|)$;

(2) (logical coverage): $\forall i \in [1\,..\,n']$ $(\|\bar{N}_i\| \supseteq \|N'_i\|)$;

(3) (temporal coverage): For each path $\bar{N}_i \rightsquigarrow \bar{N}_j = \langle \bar{N}_i, N^\epsilon_1, \ldots, N^\epsilon_s, \bar{N}_j \rangle$ in $\mathcal{O}$, where $\bar{N}_i \in \bar{\mathcal{N}}$, $\bar{N}_j \in \bar{\mathcal{N}}$, $s \geq 0$, $\forall k \in [1\,..\,s]$ $\left(N^\epsilon_k \in \mathcal{N}^\epsilon\right)$, the following holds in $\mathcal{O}'$: $N'_i \prec N'_j$.

Example 5. With reference to the observations displayed in Fig. 1, it is easy to show that $\mathcal{O}_1 \trianglerighteq \mathcal{O}_2$. Assume the subset of $\mathcal{N}_1$ being $\bar{\mathcal{N}}_1 = \{N_1, N_2, N_4, N_5\}$. Hence, $\mathcal{N}^\epsilon_1 = \{N_3\}$. Clearly, ϵ-coverage holds, as $\epsilon \in \|N_3\|$. Logical coverage holds too, as $\|N_2\| \supseteq \|N'_1\|$, $\|N_1\| \supseteq \|N'_2\|$, $\|N_4\| \supseteq \|N'_3\|$, and $\|N_5\| \supseteq \|N'_4\|$. It is easy to check that temporal coverage occurs. For instance, for $\langle N_1, N_3, N_5 \rangle$, where $N_3 \in \mathcal{N}^\epsilon_1$, we have $N'_2 \prec N'_4$ in $\mathcal{O}_2$.

Theorem 2 and Note 1 offer evidence that coverage is only sufficient for subsumption, not necessary. However, in practice, if coverage does not hold, it is unlikely for subsumption

to hold. Note that, since coverage entails subsumption, the conditions in Theorem 1 are necessary for coverage too.

Theorem 2. *Coverage entails subsumption:*

$$\mathcal{O} \trianglerighteq \mathcal{O}' \implies \mathcal{O} \ni \mathcal{O}'. \quad (9)$$

Proof. The proof is based on Definition 1 and Definition 2 (the latter given below).

Definition 2. (Sterile sequence) *Let* $\tilde{\mathcal{N}} = \langle \tilde{N}_1, \ldots, \tilde{N}_{n'} \rangle$ *be an ordering*[4] *of nodes in* $\bar{\mathcal{N}}$. *The* sterile sequence *of* $\tilde{\mathcal{N}}$,

$$\left\langle \mathcal{N}_0^\epsilon, \mathcal{N}_1^\epsilon, \ldots, \mathcal{N}_{n'}^\epsilon \right\rangle \quad (10)$$

is a sequence of subsets of $\mathcal{N}^\epsilon$, *called* sterile sets, *inductively defined as follows:*

- (Basis) $\mathcal{N}_0^\epsilon$ *is defined by the following two rules:*

(1) If $N \in \mathcal{N}^\epsilon$, N *is a root of* $\mathcal{O}$, *then* $N \in \mathcal{N}_0^\epsilon$,

(2) If $N \in \mathcal{N}^\epsilon$, *all parents of* N *are in* $\mathcal{N}_0^\epsilon$, *then* $N \in \mathcal{N}_0^\epsilon$;

- (Induction) *Given* $\mathcal{N}_i^\epsilon$, $i \in [0\,..\,(n'-1)]$, *the successive sterile set* $\mathcal{N}_{i+1}^\epsilon$ *is defined by the following two rules:*

(3) If $N \in \mathcal{N}^\epsilon$, *all parents of* N *are in* $\left(\mathcal{N}_i^* \cup \left\{\tilde{N}_{i+1}\right\}\right)$, *then* $N \in \mathcal{N}_{i+1}^\epsilon$,

(4) If $N \in \mathcal{N}^\epsilon$, *all parents of* N *are in* $\left(\mathcal{N}_i^* \cup \left\{\tilde{N}_{i+1}\right\} \cup \mathcal{N}_{i+1}^\epsilon\right)$, *then* $N \in \mathcal{N}_{i+1}^\epsilon$,

where $\mathcal{N}_i^*$, $i \in [0\,..\,n']$, *is recursively defined as follows:*

$$\mathcal{N}_i^* = \begin{cases} \mathcal{N}_0^\epsilon & \text{if } i = 0 \\ \mathcal{N}_{i-1}^* \cup \left\{\tilde{N}_i\right\} \cup \mathcal{N}_i^\epsilon & \text{otherwise.} \end{cases} \quad (11)$$

To prove the theorem, it suffices to show that each candidate signature $\mathbb{S}$ in the index space of $\mathcal{O}'$ is also a candidate signature in the index space of $\mathcal{O}$, namely:

$$\forall \mathbb{S} \in \|Isp(\mathcal{O}')\| \; (\mathbb{S} \in \|Isp(\mathcal{O})\|). \quad (12)$$

According to Theorem 1 in (Lamperti & Zanella 2002), $\mathbb{S}$ is the sequence of labels obtained by selecting, without violating the precedence constraints of $\mathcal{A}'$, one label from each node in $\mathcal{N}'$, and by removing all the null labels ϵ. Let

$$\mathbb{N}' = \langle \tilde{N}'_1, \ldots, \tilde{N}'_{n'} \rangle \quad (13)$$

be the ordering of $\mathcal{N}'$ relevant to the choices of such labels. Accordingly, the sequence $\mathbb{L}'$ of the chosen labels can be written as

$$\mathbb{L}' = \langle \ell \mid \ell \in \|\tilde{N}'_i\|, i \in [1\,..\,n'] \rangle \quad (14)$$

while the candidate signature $\mathbb{S}$ is in fact

$$\mathbb{S} = \langle \ell \mid \ell \in \mathbb{L}', \ell \neq \epsilon \rangle. \quad (15)$$

We need to show that there exists an ordering $\mathbb{N}$ of $\mathcal{N}$ fulfilling the precedence constraints imposed by $\mathcal{A}$, from which it is possible to select a sequence $\mathbb{L}$ of labels,

$$\mathbb{L} = \langle \ell_1, \ell_2, \ldots, \ell_n \rangle \quad (16)$$

such that the subsequence of non-null labels in $\mathbb{L}$ equals $\mathbb{S}$:

$$\langle \ell \mid \ell \in \mathbb{L}, \ell \neq \epsilon \rangle = \mathbb{S}. \quad (17)$$

[4] An ordering of a set is a sequence involving all and only the elements in the set, without duplicates.

Note how $\mathbb{N}$ (as well as any other ordering of $\mathcal{N}$) can be represented as a sequence of nodes in $\tilde{\mathcal{N}}$, with each node being interspersed with (possibly empty) subsequences $\mathbb{N}_i^\epsilon$ of nodes in $\mathcal{N}^\epsilon$, specifically

$$\mathbb{N} = \mathbb{N}_0^\epsilon \cup \langle \tilde{N}_1 \rangle \cup \mathbb{N}_1^\epsilon \cup \langle \tilde{N}_2 \rangle \cup \mathbb{N}_2^\epsilon \ldots \langle \tilde{N}_{n'} \rangle \cup \mathbb{N}_{n'}^\epsilon \quad (18)$$

where

$$\bigcup_{i=1}^{n'} \{\tilde{N}_i\} = \bar{\mathcal{N}}, \quad \bigcup_{i=0}^{n'} \mathbb{N}_i^\epsilon = \mathcal{N}^\epsilon, \quad \bigcap_{i=0}^{n'} \mathbb{N}_i^\epsilon = \emptyset. \quad (19)$$

The proof is by induction on $\mathbb{L}'$. Let $\mathbb{L}'_i$ denote the subsequence of $\mathbb{L}'$ up to the i-th label, $i \in [1\,..\,n']$. Let $\mathbb{L}_i$ denote the subsequence of $\mathbb{L}$ relevant to the choices of labels performed in correspondence of the labels in $\mathbb{L}'_i$. Let $\mathbb{S}_i$ and $\mathbb{S}'_i$ denote the candidate signatures corresponding to $\mathbb{L}_i$ and $\mathbb{L}'_i$, respectively.[5]

(*Basis*) No label is chosen in $\mathcal{O}'$, that is, $\mathbb{L}'_0 = \langle\rangle$. We choose a sequence of empty labels for all the nodes in $\mathcal{N}_0^\epsilon$, which is clearly possible according to the property that $\mathcal{N}_0^\epsilon$ is a sterile set composed of nodes having ancestors in $\mathcal{N}^\epsilon$ only. In other words, $\mathbb{N}_0^\epsilon$ is an ordering of $\mathcal{N}_0^\epsilon$, while $\mathbb{L}_0 = \langle \epsilon, \epsilon, \ldots, \epsilon \rangle$, hence, $\mathbb{S}_0 = \mathbb{S}'_0 = \langle\rangle$.

(*Induction*) We assume that $\mathbb{L}_i$ and $\mathbb{L}'_i$, $i \in [0\,..\,(n'-1)]$, are such that $\mathbb{S}_i = \mathbb{S}'_i$. We also assume that, given the sequence $\langle \tilde{N}'_1, \ldots, \tilde{N}'_i \rangle$ of chosen nodes in $\mathcal{O}'$, the corresponding sequence of chosen nodes in $\mathcal{O}$ is $\mathbb{N}_0^\epsilon \cup \langle \tilde{N}_1 \rangle \cup \mathbb{N}_1^\epsilon \cup \langle \tilde{N}_2 \rangle \cup \mathbb{N}_2^\epsilon \ldots \langle \tilde{N}_i \rangle \cup \mathbb{N}_i^\epsilon$, where, $\forall k \in [1\,..\,i]$, if $\tilde{N}'_k$ is the node N'_h in $\mathcal{N}'$, then $\tilde{N}_k$ is the node $\bar{N}_h$ in $\bar{\mathcal{N}}$, and each $\mathbb{N}_k^\epsilon$ is an ordering of $\mathcal{N}_k^\epsilon$. We have to show that, once chosen the next label $\ell \in \|\tilde{N}'_{i+1}\|$, thereby determining $\mathbb{L}'_{i+1}$ and $\mathbb{S}'_{i+1}$, it is possible to choose a node $\tilde{N}_{i+1} \in \bar{\mathcal{N}}$ that includes ℓ, and $\mathbb{N}_{i+1}^\epsilon$ as an ordering of $\mathcal{N}_{i+1}^\epsilon$ from which ϵ is chosen, thereby determining $\mathbb{L}_{i+1}$ such that $\mathbb{S}_{i+1} = \mathbb{S}'_{i+1}$.

Let N'_j be the node in $\mathcal{N}' = \{N'_1, \ldots, N'_{n'}\}$ corresponding to $\tilde{N}'_{i+1}$. According to logical coverage in Definition 1, there exists a node $\bar{N}_j$ in $\bar{\mathcal{N}} = \{\bar{N}_1, \ldots, \bar{N}_{n'}\}$ such that $\|\bar{N}_j\| \supseteq \|\bar{N}'_j\|$, in other words, $\bar{N}_j$ includes ℓ. We consider $\tilde{N}_{i+1} = N_j$. In order for $\bar{N}_j$ to be actually chosen, we have to show that each parent node N of $\bar{N}_j$ in $\mathcal{O}$ was already considered, that is, N belongs to the prefix of $\tilde{\mathcal{N}}$ relevant to $\mathbb{L}_i$. Two cases are possible for N:

(*a*) N is a node $\bar{N}_k \in \bar{\mathcal{N}}$. On the one hand, owing to temporal coverage, $\bar{N}_k \mapsto \bar{N}_j$ in $\mathcal{O}$ entails $N'_k \prec N'_j$ in $\mathcal{O}'$. On the other, since N'_j was chosen in $\mathcal{O}'$, all its parent nodes must have been considered already, that is, $N'_k \in \langle \tilde{N}'_1, \ldots, \tilde{N}'_i \rangle$. Since, based on the assumption of *Induction*, we always choose for each node in $N'_m \in \mathcal{N}'$ the corresponding node

[5] Note how $\mathbb{L}'_i$ includes exactly i labels, while, owing to the ϵ selected for nodes in $\mathcal{N}_\epsilon$, the number of labels in $\mathbb{L}_i$ is possibly greater than i.

$\bar{N}_m \in \bar{\mathcal{N}}$, it is possible to claim that $\bar{N}_k$ was already considered in $\mathcal{O}$, that is, $\bar{N}_k \in \langle \tilde{N}_1, \ldots, \tilde{N}_i \rangle$.

(b) $N \in \mathcal{N}_\epsilon$. We consider each path $N_\mathrm{a} \rightsquigarrow N$ in $\mathcal{O}$ such that N_a is the first ancestor of N (possibly N itself), where either N_a is a root of $\mathcal{O}$ or $N_\mathrm{a} \in \bar{\mathcal{N}}$. Let $\mathcal{N}_\mathrm{a}$ be the set of such ancestors. We show that each node $N_\mathrm{a} \in \mathcal{N}_\mathrm{a}$ has been considered already. Two cases are possible: either $N_\mathrm{a} \in \mathcal{N}_\epsilon$ or $N_\mathrm{a} \in \bar{\mathcal{N}}$. In the first case, N_a is a node in the sterile set $\mathcal{N}_0^\epsilon$ and, hence, it has been considered in $\mathbb{N}_0^\epsilon$ already (see *Basis*). In the second case ($N_\mathrm{a} \in \bar{\mathcal{N}}$), let $\bar{N}_h$ be the node in $\bar{\mathcal{N}}$ corresponding to N_a. We consider a path $\bar{N}_h \rightsquigarrow N \mapsto \bar{N}_j$. Since between $\bar{N}_h$ and $\bar{N}_j$ are only nodes in $\mathcal{N}_\epsilon$, temporal coverage implies that $N'_h \prec N'_j$ in $\mathcal{O}'$, where N'_h is the node in $\mathcal{N}'$ corresponding to $\bar{N}_h$. Thus, N'_h was already considered in $\mathcal{O}'$. As, based on the assumption in *Induction*, we always choose in $\mathcal{O}$ the corresponding node of that chosen in $\mathcal{O}'$, this implies that $\bar{N}_h$ was already considered in $\mathcal{O}$ too. We conclude that all nodes in $\mathcal{N}_\mathrm{a}$ have been considered. Now, it is clear that N is either in $\mathcal{N}_\mathrm{a}$ or N is a node belonging to the sterile set of some node in $\mathcal{N}_\mathrm{a}$. In either case, owing to the assumption of *Induction*, N must have been considered already. In other words, all parents of $\bar{N}_j$ have been chosen already, thereby allowing $\bar{N}_j$ itself, alias $\tilde{N}_{i+1}$, to be chosen. Furthermore, based on the definition of sterile sequence, we may also consider an ordering $\mathbb{N}_{i+1}^\epsilon$ of $\mathcal{N}_{i+1}^\epsilon$ and choose label ϵ for each of such nodes, thereby leading to the conclusion that $\mathbb{S}_{i+1} = \mathbb{S}'_{i+1}$. $\square$

Note 1. *Coverage is stronger than subsumption, namely:*

$$\mathcal{O} \ni \mathcal{O}' \quad \not\Rightarrow \quad \mathcal{O} \trianglerighteq \mathcal{O}'. \tag{20}$$

To be convinced, it suffices to show an example in which subsumption holds whilst coverage does not. Consider two observations, $\mathcal{O} = (\mathcal{N}, \mathcal{L}, \mathcal{A})$ and $\mathcal{O}' = (\mathcal{N}', \mathcal{L}', \mathcal{A}')$, where $\mathcal{N} = \{N_1, N_2\}$, $\mathcal{N}' = \{N'_1, N'_2\}$, $\mathcal{L} = \mathcal{L}' = \{a\}$, $\mathcal{A} = \{N_1 \mapsto N_2\}$, $\mathcal{A}' = \emptyset$, and $\|\bar{N}_1\| = \|N_2\| = \|N'_1\| = \|N'_2\| = \{a\}$, as displayed in Fig. 3.

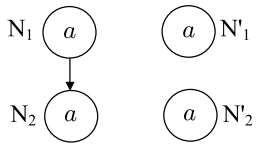


Figure 3: Observations $\mathcal{O}$ (left) and $\mathcal{O}'$ (right).

Clearly, $\bar{\mathcal{N}} = \mathcal{N}$ and $\mathcal{N}_\epsilon = \emptyset$. Note how, unlike $\mathcal{O}$, since $\mathcal{A}' = \emptyset$, $\mathcal{O}'$ does not force any temporal constraint between N_1 and N_2. Incidentally, both observations involve just one candidate signature, namely $\mathbb{S} = \langle a, a \rangle$. Thus, since $\|Isp(\mathcal{O})\| = \|Isp(\mathcal{O}')\| = \{\langle a, a \rangle\}$, both observations subsume each other, in particular $\mathcal{O} \ni \mathcal{O}'$. However, it is easy to realize that O does *not* cover $\mathcal{O}'$, namely $\mathcal{O} \not\trianglerighteq \mathcal{O}'$. In fact, due to the symmetry of $\mathcal{O}'$, we can choose any of the two possible associations between nodes in $\mathcal{O}$ and nodes in $\mathcal{O}'$, for instance, $\bar{\mathcal{N}} = \{N_1, N_2\}$. Based on Definition 1, on the one hand, both ϵ-coverage and logical coverage hold. On the other, temporal coverage is missing, as for $N_1 \mapsto N_2$ in $\mathcal{O}$, we have $N'_1 \not\prec N'_2$. The same negative result occurs for $\bar{\mathcal{N}} = \{N_2, N_1\}$. In other terms, $\mathcal{O} \not\trianglerighteq \mathcal{O}'$.

Testing Coverage

In this section we give an abstract, pseudo-coded implementation of subsumption-checking via coverage. Specifically, Algorithm 1 tests both the necessary conditions of Theorem 1 and the coverage relationship. A tracing of the algorithm on observations in Fig. 1 is provided in Example 6.

Algorithm 1. (COVERS) The *Covers* function (lines 1–41) takes as input two observations, $\mathcal{O}$ and $\mathcal{O}'$, and outputs a Boolean value indicating whether or not $\mathcal{O}$ covers $\mathcal{O}'$. The body of *Covers* is outlined in lines 30–41. In lines 31–32, the observation parameters for $\mathcal{O}$ and $\mathcal{O}'$ are set. Then, at line 33, conditions (5) and (6) of Theorem 1, along with condition (8) of Corollary 1.1, are checked. In lines 36–38, the multisets $\mathcal{M}$ and $\mathcal{M}'$ of instances of labels are created, with the former decremented by $d = (n - n')$ instances of label ϵ, which is the cardinality of $(\mathcal{N} - \mathcal{N}')$. This allows the algorithm to retain a sufficient number of spare nodes in $\mathcal{N}$ that contain ϵ, namely $\mathcal{N}^\epsilon$ in Definition 1. At line 39, condition (7) of Theorem 1 is checked. The algorithm yields $\bar{\mathcal{N}}$, the subset of $\mathcal{N}$ that is associated with $\mathcal{N}'$ in Definition 1, by building the set $\mathcal{R}$ of associations through the call to the auxiliary function *CovStep* at line 40. The specification of *CovStep* is given in lines 3–29. Besides $\mathcal{O}$, $\mathcal{O}'$, $\mathcal{M}$, and $\mathcal{M}'$, it takes as input $\mathcal{C}$ and $\mathcal{C}'$, the set of nodes already considered in $\mathcal{O}$ and $\mathcal{O}'$, respectively, along with d, the number of nodes in $\mathcal{N}^\epsilon$ not yet considered, and $\mathcal{R}$, the set of associations made up so far. The body of *CovStep* starts at line 10, where the cardinality of $\mathcal{R}$ is tested: if $\mathcal{R}$ contains n' pairs, it means that all nodes in $\mathcal{N}'$ have been considered and $\bar{\mathcal{N}}$ is completed, thereby, coverage holds. Otherwise, a new node N' in $\mathcal{O}'$ is considered at line 11, such that all its parent nodes have been considered already. At line 12, the set $\mathcal{F}$ of nodes in $\mathcal{O}$ is created, which includes the unconsidered nodes of $\mathcal{O}$ with all parents already in $\mathcal{C}$. A loop for each node N in $\mathcal{F}$ is iterated in lines 13–27. First, logical coverage and containment relationship of labels are tested (line 14). Then, the set $\mathcal{N}_a$ of the nearest ancestors of N which have been already involved in the associations of $\mathcal{R}$ is instantiated (line 15). This allows temporal-coverage checking (line 16). If the latter succeeds, *CovStep* is recursively called at line 17, with new actual parameters: the sets $\mathcal{C}$ and $\mathcal{C}'$ of considered nodes are extended with N and N', respectively, the multisets $\mathcal{M}$ and $\mathcal{M}'$ are decremented by the labels in N and N', respectively, while $\mathcal{R}$ is extended with the new pair (N, N'). If such a call succeeds, the current activation of *CovStep* succeeds too (line 18). If not, or either logical or temporal coverage fails, a chance still remains by assuming $N \in \mathcal{N}^\epsilon$: this is viable only on condition that N include ϵ, there exists at least one spare node in $\mathcal{N}^\epsilon$ ($d > 0$), and the multiset $\mathcal{M}$ contains $\mathcal{M}'$ once decremented by the labels of N (line 22). If so, a different recursive call to *CovStep* is performed at line 23, with the changed parameters being the (extended) set $\mathcal{C}$ of consumed nodes in $\mathcal{O}$, the (decremented) multiset $\mathcal{M}$, and the decremented value of d. If

such a call succeeds, the current activation of *CovStep* succeeds too. If not, the loop is iterated and a new node in $\mathcal{F}$ is tried. Clearly, if the computation exits the loop in a natural way, it means that no node can be associated with N' within this computational context, thereby causing the current activation of *CovStep* to fail (line 28).

```
1.  function Covers(O, O'): Boolean
2.   O = (N, L, A), O' = (N', L', A'): two temporal observations;
3.   function CovStep(O, O', C, C', M, M', d, R): Boolean
4.    O = (N, L, A), O' = (N', L', A'): two temporal observations,
5.    C, C': the set of consumed nodes for O and O', respectively,
6.    M, M': the multisets of labels in O and O', respectively,
7.    d: the number of nodes remaining in N that can still be in N^ε,
8.    R ⊆ N × N': a relation on N and N';
9.   begin {CovStep}
10.   if |R| = n' then return true end-if;
11.   Pick up a node N' ∈ (N' − C') where all parents of N' are in C';
12.   F := the set of N ∈ (N − C) where all parents of N are in C;
13.   for each N ∈ F do
14.    if ‖N‖ ⊇ ‖N'‖ and (M − ‖N‖) ⊇ (M' − ‖N'‖) then
15.     N_a := the set of nearest ancestors of N which are in R(N);
16.     if ∀N_a ∈ N_a, (N_a, N'_a) ∈ R (N'_a ≺ N') then
17.      if CovStep(O, O', C ∪ {N}, C' ∪ {N'}, M − ‖N‖,
                                    M' − ‖N'‖, d, R ∪ {(N, N')}) then
18.       return true
19.      end-if
20.     end-if
21.    end-if;
22.    if ε ∈ ‖N‖ and d > 0 and (M − ‖N‖) ⊇ M' then
23.     if CovStep(O, O', C ∪ {N}, C',
                                M − ‖N‖, M', d − 1, R) then
24.      return true
25.     end-if
26.    end-if
27.   end-for;
28.   return false
29.  end {CovStep};
30.  begin{Covers}
31.  n := |N|; n_ε := {N | N ∈ N, ε ∈ ‖N‖};
32.  n' := |N'|; n'_ε := {N' | N' ∈ N', ε ∈ ‖N'‖};
33.  if n < n' or n_ε − n'_ε < n − n' or L ⋣ L' then
34.   return false
35.  end-if:
36.  Create the multisets M and M' of instances of labels in O, O';
37.  d := n − n';
38.  Remove d instances of label ε from M;
39.  if M ⋣ M' then return false end-if;
40.  return CovStep(O, O', ∅, ∅, M, M', d, ∅)
41. end {Covers}.
```

Example 6. With reference to the observations in Fig. 1, consider the run of $Covers(\mathcal{O}_1, \mathcal{O}_2)$. Since, according to Example 4, all the necessary conditions of Theorem 1 hold, we focus our attention on the first call to *CovStep* at line 40.

Depicted in Fig. 4 is the tree of the recursive activations to *CovStep*, where each node i corresponds to the i-th call (dashed nodes correspond to calls at line 23, with the others corresponding to line 17). Relevant details are given in Table 1, with *Id* being the identifier of the call, while the other columns indicate the actual parameters of the call (observation nodes are identified by the corresponding subscripts). The computation is described by the following steps, where item numbers stand for activation identifiers, namely *Id*.

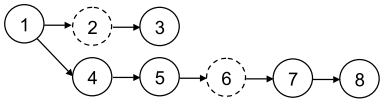


Figure 4: Activation tree for *CovStep* in Example 6.

1. $N' = 1'$, $\mathcal{F} = \{1, 2\}$. Within the loop (line 13), choosing $N = 1$ makes the multiset containment false (line 14). However, since condition at line 22 holds for N, a recursive call to *CovStep* is performed at line 23 (see $Id = 2$ in Table 1).
2. $N' = 1'$, $\mathcal{F} = \{2, 3\}$. With $N = 2$, a recursive call is performed at line 17 ($Id = 3$ in Table 1).
3. $N' = 2'$, $\mathcal{F} = \{3, 4\}$. With $N = 3$, logical coverage fails, as $\|N\| \not\supseteq \|N'\|$. Besides, although $\epsilon \in \|N\|$, condition at line 22 is false because $d = 0$ (no further spare nodes to assume in $\mathcal{N}^\epsilon$). Thus, the control returns to the second call: with $N = 3$, logical coverage fails, while condition at line 22 is false since $d = 0$. This causes the control to return to the first call, where $N = 2$ is chosen: this allows the fourth recursive call at line 17 ($Id = 4$).
4. $N' = 2'$, $\mathcal{F} = \{1, 4\}$. With $N = 1$, a recursive call is performed at line 17 ($Id = 5$).
5. $N' = 3'$, $\mathcal{F} = \{3, 4\}$. With $N = 3$, logical coverage fails. However, since condition at line 22 holds, a recursive call is performed at line 23 ($Id = 6$).
6. $N' = 3'$, $\mathcal{F} = \{4\}$. With $N = 4$, a recursive call is performed at line 17 ($Id = 7$).
7. $N' = 4'$, $\mathcal{F} = \{5\}$. With $N = 5$, a recursive call is performed at line 17 ($Id = 8$).
8. At line 10, since $|\mathcal{R}| = 4$, *CovStep* succeeds.

Proposition 1. *Algorithm 1 is a sound and complete implementation of coverage:*

$$Covers(\mathcal{O}, \mathcal{O}') \iff \mathcal{O} \trianglerighteq \mathcal{O}'. \tag{21}$$

Proof (sketch). To prove equivalence (21), we first show

$$Covers(\mathcal{O}, \mathcal{O}') \implies \mathcal{O} \trianglerighteq \mathcal{O}'. \tag{22}$$

Assuming $Covers(\mathcal{O}, \mathcal{O}')$ succeeding means that the call to *CovStep* at line 40 returns **true**. Function *CovStep* recursively instantiates the set $\mathcal{R}$ of associations of nodes (N, N'), for which both logical coverage (line 14) and temporal coverage (line 16), required by Definition 1, hold. Moreover, ϵ-coverage is supported by conditions at line 22 and the initialization at lines 36–38, which allow for retaining the $(n - n')$ nodes of $\mathcal{N}^\epsilon$ once $\mathcal{R}$ is completed (line 10). In other words, entailment (22) holds. Then, we have to show

$$\mathcal{O} \trianglerighteq \mathcal{O}' \implies Covers(\mathcal{O}, \mathcal{O}'). \tag{23}$$

The proof is by contradiction. Assume that $\mathcal{O} \trianglerighteq \mathcal{O}'$, while $Covers(\mathcal{O}, \mathcal{O}') =$ **false**. Based on Definition 1, let $\mathcal{R}^* = \{(\bar{N}_1, N'_1), \ldots, (\bar{N}_{n'}, N'_{n'})\}$ denote the relation between $\bar{\mathcal{N}}$ and $\mathcal{N}'$. Based on a run of *Covers*, we show that *Covers* necessarily makes up $\mathcal{R} = \mathcal{R}^*$. The proof is by induction on $\mathcal{R}$. Note how we can restrict our analysis

Table 1: Tracing of $Covers(\mathcal{O}_1, \mathcal{O}_2)$ in Example 6.

Id	$\mathcal{C}$	$\mathcal{C}'$	$\mathcal{M}$	$\mathcal{M}'$	d	$\mathcal{R}$
1	$\emptyset$	$\emptyset$	$\{a,a,a,b,b,b,b,c,d,f,\epsilon,\epsilon\}$	$\{a,a,b,c,d,\epsilon,\epsilon\}$	1	$\emptyset$
2	$\{1\}$	$\emptyset$	$\{a,a,b,b,b,c,d,f,\epsilon,\epsilon\}$	$\{a,a,b,c,d,\epsilon,\epsilon\}$	0	$\emptyset$
3	$\{1,2\}$	$\{1'\}$	$\{a,b,b,c,d,f,\epsilon,\epsilon\}$	$\{a,b,c,d,\epsilon,\epsilon\}$	0	$\{(2,1')\}$
4	$\{2\}$	$\{1'\}$	$\{a,a,b,b,b,c,d,f,\epsilon,\epsilon\}$	$\{a,b,c,d,\epsilon,\epsilon\}$	1	$\{(2,1')\}$
5	$\{1,2\}$	$\{1',2'\}$	$\{a,b,b,c,d,f,\epsilon\}$	$\{a,c,d,\epsilon\}$	1	$\{(2,1'),(1,2')\}$
6	$\{1,2,3\}$	$\{1',2'\}$	$\{a,b,b,c,d,\epsilon\}$	$\{a,c,d,\epsilon\}$	0	$\{(2,1'),(1,2')\}$
7	$\{1,2,3,4\}$	$\{1',2',3'\}$	$\{a,b,\epsilon\}$	$\{a,\epsilon\}$	0	$\{(2,1'),(1,2'),(4,3')\}$
8	$\{1,2,3,4,5\}$	$\{1',2',3',4'\}$	$\emptyset$	$\emptyset$	0	$\{(2,1'),(1,2'),(4,3'),(5,4')\}$

to the recursive call to *CovStep*, as lines 31–39 simply check the necessary conditions of subsumption stated by Theorem 1 and Corollary 1.1. In fact, since coverage entails subsumption (Theorem 2), such conditions are necessary for coverage too, in other words, the computation surely reaches the call to *CovStep* at line 40. Moreover, such call is supposed to return **false** (owing to the assumption $Covers(\mathcal{O}, \mathcal{O}') = \textbf{false}$).

(*Basis*) Focus on the first call to *CovStep*, where $\mathcal{C}$, $\mathcal{C}'$, and $\mathcal{R}$ are empty, and consider the (first) node N' chosen at line 11. Let N' correspond to the j-th node in $\mathcal{N}'$, namely N'_j. Let $\bar{N}_j$ be the node in $\bar{\mathcal{N}}$ associated with N'_j in $\mathcal{R}^*$, namely $(\bar{N}_j, N'_j) \in \mathcal{R}^*$. Based on Definition 1, temporal coverage requires that, for each path $\bar{N}_i \rightsquigarrow \bar{N}_j$ in $\mathcal{O}$, where all intermediate nodes in the path are in $\mathcal{N}^\epsilon$, we have $N'_i \prec N'_j$ in $\mathcal{O}'$. However, none of such paths exists, as $N' = N'_j$ is chosen with $\mathcal{C}' = \emptyset$, that is, N' has no parent nodes. Consequently, all ancestors of $\bar{N}_j$ in $\mathcal{O}$ (if any) are in $\mathcal{N}^\epsilon$, that is, they contain label ϵ. Since *CovStep* is supposed to fail, it will try all choices of N in $\mathcal{F}$. Two cases are possible: either $\bar{N}_j$ is a root of $\mathcal{O}$ or all ancestors of $\bar{N}_j$ are in $\mathcal{N}^\epsilon$. In the first case, $N = \bar{N}_j$ is associated in $\mathcal{R}$ with $N' = N'_j$ within the recursive call to *CovStep* at line 17. In the second case, the same association will be created after a number of recursive calls of *CovStep* at line 23, as all calls to *CovStep* are assumed to fail (including the one creating such association). Thus, in any case, the first choice of N' will led to an association (N, N') which is in $\mathcal{R}^*$ too.

(*Induction*) Assume, in the current call to *CovStep*, $\mathcal{R} = \{(\bar{N}_{c_1}, N'_{c_1}), \ldots, (\bar{N}_{c_k}, N'_{c_k})\}$, where $\mathcal{R} \subset \mathcal{R}^*$, that is, all associations yielded by *CovStep* are in $\mathcal{R}^*$ too. Let $\bar{\mathcal{N}}_c$ and $\mathcal{N}'_c$ denote the projections of $\mathcal{R}$ on $\mathcal{N}$ and $\mathcal{N}'$, respectively. Now, consider the next choice of N' at line 11. Let N' correspond to the j-th node in $\mathcal{N}'$, namely N'_j. Let $\bar{N}_j$ be the node in $\bar{\mathcal{N}}$ associated with N'_j in $\mathcal{R}^*$, namely $(\bar{N}_j, N'_j) \in \mathcal{R}^*$. Based on Definition 1, temporal coverage requires that, for each path $\bar{N}_i \rightsquigarrow \bar{N}_j$ in $\mathcal{O}$, where all intermediate nodes in the path are in $\mathcal{N}^\epsilon$, $N'_i \prec N'_j$ holds in $\mathcal{O}'$. This implies that all N'_i are in $\mathcal{N}'_c$ and, in consequence of the inductive assumption, all $\bar{N}_i$ are in $\bar{\mathcal{N}}_c$. Hence, following the same argumentation outlined in *Basis*, $\bar{N}_j$ can be considered and associated with N'_j. Therefore, $(\bar{N}_j, N'_j)$ is inserted into $\mathcal{R}$. This leads to the claim that $(\mathcal{R} \cup \{(\bar{N}_j, N'_j)\}) \subseteq \mathcal{R}^*$, which concludes the proof of *Induction*. Thus, equation (23) is proved. □

Rapid Prototyping

We prototyped the algorithms in Haskell functional language (Thompson 1999), and ran a number of experiments in order to assess the coverage approach to subsumption checking based on different classes of observations. We ran subsumption checking using two different algorithms, namely *Subsumes* and *Covers*, where the former is strictly based on the definition of subsumption and requires testing index-space containment. We considered three classes of observations, namely *disconnected*, *connected*, and *linear*. In disconnected observations, no temporal constraints are given among nodes, thereby maximizing temporal uncertainty. Instead, in connected observations, each node is temporally linked with other nodes. Linear observations are a subclass of connected observations where no temporal uncertainty occurs. The experimental results in this paper refer to connected observations. In order to stress the computation, we chose observations for which subsumption hold, so that the necessary conditions in Theorem 1 always hold.[6]

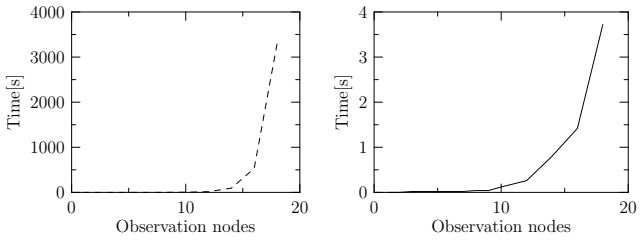


Figure 5: Checking subsumption: time response.

Shown in Fig. 5 is the response time for the two algorithms, with the x-axis marked by the number of nodes in the involved observations. Precisely, the y-axis indicates the time for *Subsumes* (dashed line, on the left) and *Covers* (plain line, on the right) to emit the relevant verdict. Considering the different scale of the y-axis, the comparison is striking in favor of *Covers*. Displayed in Fig. 6 is the maximum space allocation for the two algorithms, which shows how no considerable difference exists between them.

[6] In fact, when one of such conditions is violated, *Covers* is increasingly more efficient than *Subsumes*.

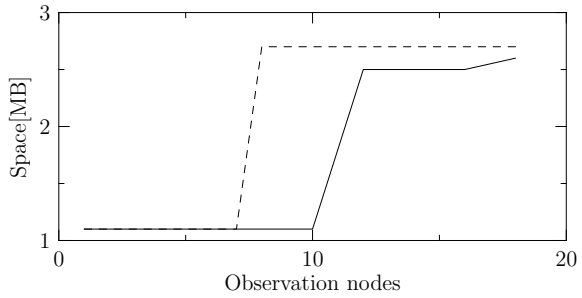


Figure 6: Checking subsumption: space allocation.

Conclusion

A technique for checking observation-subsumption in diagnosis of DESs has been proposed. This check is required to pursue similarity-based diagnosis, where the solution of a diagnostic problem is possibly supported by the solution of a previously-solved problem stored in a knowledge base. In the literature, the solution to such checking-problem is strictly performed based on the definition of observation-subsumption, which requires the generation and comparison of the index spaces of the two observations. Since index-space generation and manipulation are computationally complex, an alternative technique has been envisaged and formally defined in this paper, which exploits a number of necessary conditions, as well as a sufficient condition, for subsumption to hold. The latter is based on the notion of coverage, which allows the direct comparison of the two observations without any index-space generation or manipulation. The new approach has been tested and compared with the systematic approach. Experimental results indicate that the technique is considerably worthwhile as to time complexity. However, since the implementation is based on a pure functional language, chances are that implementing it through a more efficient general-purpose language is bound to still better figures.

References

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

Brand, D., and Zafiropulo, P. 1983. On communicating finite-state machines. *Journal of ACM* 30(2):323–342.

Cassandras, C., and Lafortune, S. 1999. *Introduction to Discrete Event Systems*, volume 11 of *The Kluwer International Series in Discrete Event Dynamic Systems*. Boston, MA: Kluwer Academic Publisher.

Cerutti, S.; Lamperti, G.; Scaroni, M.; Zanella, M.; and Zanni, D. 2007. A diagnostic environment for automaton networks. *Software – Practice and Experience* 37(4):365–415. DOI: 10.1002/spe.773.

Chen, Y., and Provan, G. 1997. Modeling and diagnosis of timed discrete event systems - a factory automation example. In *American Control Conference*, 31–36.

Console, L.; Picardi, C.; and Ribaudo, M. 2002. Process algebras for systems diagnosis. *Artificial Intelligence* 142(1):19–51.

Debouk, R.; Lafortune, S.; and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamic Systems: Theory and Application* 10:33–86.

Hopcroft, J.; Motwani, R.; and Ullman, J. 2006. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, third edition.

Lamperti, G., and Zanella, M. 2002. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence* 137(1–2):91–163.

Lamperti, G., and Zanella, M. 2003. *Diagnosis of Active Systems – Principles and Techniques*, volume 741 of *The Kluwer International Series in Engineering and Computer Science*. Dordrecht, NL: Kluwer Academic Publisher.

Lamperti, G., and Zanella, M. 2006. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence* 170(3):232–297.

Lunze, J. 2000. Diagnosis of quantized systems based on a timed discrete-event model. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 30(3):322–335.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence* 164:121–170.

Pencolé, Y. 2000. Decentralized diagnoser approach: application to telecommunication networks. In *Eleventh International Workshop on Principles of Diagnosis – DX'00*, 185–192.

Rozé, L., and Cordier, M. 2002. Diagnosing discrete-event systems: extending the 'diagnoser approach' to deal with telecommunication networks. *Journal of Discrete Event Dynamic Systems: Theory and Application* 12:43–81.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology* 4(2):105–124.

Sampath, M.; Lafortune, S.; and Teneketzis, D. 1998. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control* 43(7):908–929.

Schullerus, G., and Krebs, V. 2001. Diagnosis of a class of discrete-event systems based on parameter estimation of a modular algebraic model. In *Twelfth International Workshop on Principles of Diagnosis – DX'01*, 189–196.

Thompson, S. 1999. *Haskell – The Craft of Functional Programming*. Harlow, UK: Addison-Wesley.

Zad, S.; Kwong, R.; and Wonham, W. 1999. Fault diagnosis in timed discrete-event systems. In *38th IEEE Conference on Decision and Control – CDC'99*, 1756–1761. Pheonix, AZ: IEEE, Piscataway, NJ.

Inversion-based residual generation for robust detection and isolation of faults by means of estimation of the inverse dynamics in linear dynamical systems*

A. Edelmayer,† J. Bokor, Z. Szabó

Systems and Control Laboratory
Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende u. 13-17, H-1111, Budapest
Hungary

S. Molnár

Department of Informatics
Faculty of Mechanical Engineering
St. István University
Páter K. u. 1, H-2103, Gödöllő
Hungary

Abstract

In this paper the idea of inversion-based direct input reconstruction for robust detection, separation and estimation of multiple simultaneous faults in the presence of persistent disturbances in linear dynamical systems is presented. In particular, it is shown how in a specific filtering structure a residual generator, relying on the inverse representation of the system, by means of estimation of the inverse dynamics can be designed providing robust fault decoupling. The development of the method and its applicability to earlier not solvable problems is demonstrated based on an aircraft monitoring application example.

INTRODUCTION

The distinctive term *direct input reconstruction* is used to identify a relatively new idea that has been proposed and investigated in the methodology of the design of detection filters for detection, separation and estimation of faults in both linear and nonlinear systems quite lately. This approach is an application of dynamic inversion to filtering which is dual to the concept of dynamic inversion for control. The difference between these inversion approaches is that control uses a right inverse whereas estimation uses a left inverse of the system.

The method arrives at detector architectures whose residual outputs consist of the fault signals while their inputs are the standard observables (inputs and outputs) of the system and possible their time derivatives. This makes not only the detection and isolation but also the estimation of the fault signals possible as it was rendered in algebraic (Szigeti, Bokor, & Edelmayer 2002) and geometric approaches (Edelmayer, Vera, & Szigeti 2002; Edelmayer *et al.* 2004), respectively.

Based on the recent results of (Edelmayer, Bokor, & Szabó 2006), in this paper, the effectiveness and distinctive capabilities of the direct input reconstruction method applied to robust fault detection and isolation is demonstrated. The calculation of the inverse is done in a simple algebraic way, we do not focus on properties of the inverse but on the applicability of the idea showing how advanced methods of detection filter design, such as inversion-based residual generation, H_∞ optimal filtering and advanced state estimation, and the novel combination of them may contribute to the solution of earlier not solvable problems. For more technical details of the inversion-based detection filter idea the reader is directed to the references above. For the specific filter design problem that will be pursued further in this paper, consider the linear dynamical system subject to faults and external disturbances which is given in the state space form

$$\dot{x} = Ax + B_u u + B_d d + L_2 f_2 \quad (1)$$
$$y_1 = C_1 x + D_{u,1} u + D_{d,1} d + M f_1, \quad (2)$$
$$y_2 = C_2 x + D_{u,2} u + D_{d,2} d, \quad (3)$$

where M is full rank with $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ and $y_1 \in \mathbb{R}^{p_1}$ and $y_2 \in \mathbb{R}^{p_2}$, where the rest of the matrices of the system are given in the appropriate dimensions. The unknown, bounded time functions $f_1(t), f_2(t), d(t)$ represent both actuator and sensor faults and the disturbance, respectively.

Problem 1 Assume the system subject to faults and disturbances is given as (1)-(3). Assume, as an underlying working condition, moreover, that the faults and the disturbances are coupled in the state space so that they *cannot* be separated from each other by using traditional filter design techniques such as geometric decoupling or unknown input observer design methods. Let our objective be to construct if possible a *stable* and possibly *minimal* representation of a robust residual generator in the state space that shows up the effects of $f(t)$ at its output irrespectively of the presence of the disturbance input $d(t)$ so that the separation between the respective fault transmission levels is maintained making the detection and isolation robustly possible.

Due to the presence of the non-separable disturbance in the system, the variety of applicable solution approaches available in the literature are basically confined to methods providing suppression of the disturbance effects on the filter's residual. This solution can be summarized as follows:

*This work has been supported by the Hungarian Scientific Research Fund (OTKA) under grant number K 061081, which is gratefully acknowledged by the authors.

†Corresponding author. Phone: +36 30 9 949 821, E-mail: edelmayer@sztaki.hu

Solution 1 (Robust detection with optimal disturbance attenuation). The classical solution of Problem 1 can be approached with using game theoretic or H_∞ filtering making use of one of the design methodologies presented by several authors in a number of different papers, see e.g., (Chung & Speyer 1998; Frank & Ding 1994; Edelmayer, Bokor, & Keviczky 1996; Chen & Patton 2000). By using this solution method, separation of multiple simultaneous faults relying on a single filter residual can be difficult in the practice. For the enhancement of fault selectivity of the filter, some extension of the H_∞ filtering idea have been proposed, see e.g., (Douglas & Speyer 1999; Edelmayer & Bokor 2002).

For a novel solution alternative consider the idea of direct input reconstruction which views this robust residual generator design problem as constructing the inverse of the dynamical system and use it as residual generator by following the solution idea specified below.

Solution 2 (Inversion-based fault detection and isolation by means of estimation of the inverse dynamics). Consider system (1)-(3) subject to faults and disturbances and suppose that the system from the faults to the given outputs is left invertible, i.e., one has

$$f_1 = C_{f_1}x + B_{y_1f_1}y_1 + B_{uf_1}u \tag{4}$$
$$f_2 = C_{f_2}x + B_{y_2f_2}\xi_{y_2} + B_{uf_2}\xi_u \tag{5}$$

provided that $d = 0$ and $f = [f_1\ f_2]^T$ and

$$\xi_y = [y, \dot{y}, \ldots, y^{(k-1)}]^T, \quad \xi_u = [u, \dot{u}, \ldots, u^{(k)}]^T$$

where k is the maximum relative degree of the system (1)-(3) corresponding to the pair (f_2, y_2), for details see (Edelmayer *et al.* 2004). Substituting (4)-(5) to the state equation (1) one obtains the inverse dynamics

$$\dot{x} = \bar{A}x + B_y\xi_y + B_u\xi_u + B_dd \tag{6}$$

that provides a useful structure for constructing a filter for generating detection residuals as shown in Fig. 1.

There exist many different ways of finding the inverse of linear dynamical systems, such as those, defined by the various zero dynamics algorithms, see e.g., (Isidori 1985) and (Nijmeijer & Van der Schaft 1991). In a classical approach the unknown state in (4)-(5) is expressed in terms of the derivatives of the output functions and the components of the possible zero dynamics. Since the initial conditions are not known, this approach might provide a reliable solution in a closed-loop manner only, see e.g., (Isidori 1985). As another condition of this solution, the zero dynamics of the system is required to be stable in order to obtain a stable residual generator.

Assuming, however, that additional measurements, e.g.,

$$y_3 = C_3x + D_{u,3}u + D_{d,3}d,$$

are available, such that the system with output functions $\bar{y} = [y_2, y_3]^T$ is observable, it is possible to construct a state observer for the estimation of the state x of (6). Relying on this estimation one can obtain the reconstructed fault signals by using (4)-(5).

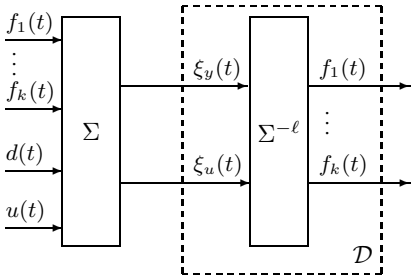


Figure 1: *Direct input reconstruction by means of system inversion: Σ is the plant, $\mathcal{D}$ is the reconstructor or detector which can be obtained as the (left) inverse $\Sigma^{-\ell}$ of the system.*

The following proposition is the main contribution of this paper which gives new aspects to the inversion-based residual generator design.

Proposition 1 Assume the system (1)-(3) is left invertible *w.r.t.* each of the unknown inputs to be detected and isolated and construct the inverse system resulting in the respective forms (4-6). Define new output functions $\bar{y} = \bar{C}x$, $\bar{y} = \{\, y_\ell \,\}|_{\ell\in\{1,\ldots,p\},j\neq\ell}$, $\dim\bar{y} < \dim y$, which, due to $\dim y > \dim f$, are not parts of the calculation of the inverse in (4)-(5), i.e., the new observations $\bar{C}$ are composed of selected rows of C. Consider the representation (4-6) equipped with the new measurement functions $\bar{y}$ and assume the pair $(\bar{A}, \bar{C})$ observable. Then, a state observer can be designed to get an estimate of the unknown state in (6). Alternatively, if the pair $(\bar{A}, \bar{C})$ is found non-observable $\bar{y}(t)$ can be extended with one or more of the original measurements $\{y_j\}$ attempting to construct an observable representation.

In the following part of this paper a comparative study of the various design alternatives associated with Solution 1 and 2 is presented. The solution alternatives are characterized on the basis of the application example patterned after the filtering problem originally raised by (Douglas & Speyer 1995) and (Chung & Speyer 1998). It is shown how Solution 1 allows disturbance attenuated non-decoupling solution by means of traditional H_∞ filtering which is brought up for reference purposes only. Then, following the idea depicted by Solution 2, two different estimation approaches of the determination of the inverse dynamics are presented yielding two distinct (approximate and exact) decoupling solutions.

THE F16XL AIRCRAFT MONITORING PROBLEM REVISITED

This example considers the design of an aircraft fault detection filter that monitors an elevon actuator and a normal accelerometer sensor in the presence of persistent wind gust disturbance where the disturbance could not be decoupled from the faults using traditional (C, A)-invariant subspace design, and could not be attenuated either while keeping fault effects decoupled. To describe aircraft dynamics a reduced-order five-state model (linearized about trimmed

level flight at $10.000\,\mathrm{ft}$ altitude ($3048\,\mathrm{m}$) and relative Mach speed 0.9) is used, representing the longitudinal dynamics only including a first-order wind gust model: the port and starboard elevons are modeled as a slaved system, no lateral dynamics and no elevon actuator dynamics are taken into consideration. This simplified aircraft model can be considered in the state space as

$$\begin{aligned} \dot{x} &= Ax + B_\omega \omega + B_\delta \delta, \\ y &= Cx + Dv, \end{aligned} \tag{7}$$

where the elevon deflection angle $\delta(t)$ is taken as the input and $\omega(t)$ and $v(t)$ stand for wind gust disturbance and sensor noise, respectively. The state and input/output variables of the system contained in system model (7) are summarized in Table I and II. Note that this wind gust model (which is usually referred to *Dryden gust model* in the literature) is a wind turbulence model recommended for study of vehicle response to winds for horizontally flying aircrafts with flight path angles less than $30\,\mathrm{deg}$. The parameters of system (7) are given by the matrices

$$A = \begin{bmatrix} -0.0674 & 0.0430 & -0.8886 & -0.5587 & 0.0430 \\ 0.0205 & -1.4666 & 16.5800 & -0.0299 & -1.4666 \\ 0.1377 & -1.6788 & -0.6819 & 0 & -1.6788 \\ 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.1948 \end{bmatrix},$$

$$B_\omega = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1.57 \end{bmatrix}, \quad B_\delta = \begin{bmatrix} -0.1672 \\ -1.5179 \\ -9.7842 \\ 0 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0.0591 & 0 & 0 & 0.0591 \\ 0.0139 & 1.0517 & 0.1485 & -0.0299 & 0 \\ -0.0677 & 0.0431 & 0.0171 & 0 & 0 \end{bmatrix},$$

$$D = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.143 & 0 & 0 \\ 0 & 0 & 0.245 & 0 \\ 0 & 0 & 0 & 0.245 \end{bmatrix}.$$

For notational convenience, let us introduce the representations $A = \begin{bmatrix} A_1^T & A_2^T & \cdots & A_5^T \end{bmatrix}^T$ and $C = \begin{bmatrix} c_1^T & c_2^T & c_3^T & c_4^T \end{bmatrix}^T$ where A_i and c_i correspond to the i-th rows of the system matrices A and C, respectively. The Dryden model (7) including wind gust disturbance is extended to include faults: a normal accelerometer sensor fault and an elevon fault as an actuator fault, denoted by $\mu_{A_z}(t)$ and $\nu_\delta(t)$, respectively. The accelerometer sensor fault $\mu_{A_z}(t)$ and the elevon fault $\nu_\delta(t)$ can be modeled as additive terms in the state and measurement equations as

$$\begin{aligned} \dot{x} &= Ax + B_\omega \omega + B_\delta(\delta + \nu_\delta), \\ y &= Cx + E_{A_z}\mu_{A_z} \end{aligned} \tag{8}$$

i.e., the elevon fault enters the system in the same direction of the state space as the input does, where $\mu_{A_z}(t), \nu_\delta(t)$ are arbitrary time-varying real scalars and, $E_{A_z} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$. Let our objective be the design of a filter for the detection and isolation of the faults in the presence of the wind gust disturbance $\omega(t)$. For simplicity, by relaxing the design assumptions, the sensor noise $v(t)$ will be considered zero in the development. The consequences of this assumption on the filtering solution will be commented later. It was (Beard 1971) followed by (White & Speyer 1987) and others, who showed that sensor faults appearing in the measurement equations can be modeled as two dimensional additive signals entering the state dynamics of the system. Based on this idea, system (8) can be converted to the state space representation

$$\begin{aligned} \dot{x} &= Ax + F_{A_z} m_{A_z} + B_\delta(\delta + \nu_\delta) + B_\omega \omega, \\ y &= Cx \end{aligned} \tag{9}$$

with $\begin{bmatrix} F_{A_z^1} & F_{A_z^2} \end{bmatrix}$, where m_{A_z} is a fictitious signal representing the sensor fault effect and the directions $F_{A_z^1}$ and $F_{A_z^2}$ are given by $E_{A_z} = CF_{A_z^1}$ and $F_{A_z^2} = AF_{A_z^1}$ so that

$$F_{A_z} = \begin{bmatrix} 0 & 0.9986 \\ 0 & 0.0534 \\ 0 & 0 \\ 1.0000 & 0 \\ 0 & 0 \end{bmatrix}.$$

It was shown in (Douglas & Speyer 1995) and (Chung & Speyer 1998) how an unobservability subspace with respect to the wind gust disturbance is formed in this application, thereby decoupling the wind gust from the fault isolation residuals, happens to be nonmutually detectable with respect to the faults by using traditional geometric decoupling methods. Moreover, the faults and the wind direction combine to place a system transmission zero at 0.0008, causing conventional detection filter design, such as unknown input observer and other geometric decoupling methods, to have an unstable closed-loop pole. For alternative solutions of the robust detection filter design problem formulated in Problem 1 the design propositions presented in accord with Solution 1 and 2 can be considered as follows.

Table 1: State Variables of the System

$x_1 = u(t)$	longitudinal body axis velocity	$\mathrm{ft/s}$
$x_2 = w(t)$	normal body axis velocity	$\mathrm{ft/s}$
$x_3 = q(t)$	pitch rate	$\mathrm{deg/s}$
$x_4 = \theta(t)$	pitch angle	deg
$x_5 = w_g(t)$	wind gust	$\mathrm{ft/s}$

Table 2: Input/Output Variables of the System

$u = \delta(t)$	elevon deflection angle	deg
$y_1 = q(t)$	pitch rate	$\mathrm{deg/s}$
$y_2 = \alpha(t)$	angle of attack	deg
$y_3 = A_z(t)$	normal acceleration	$\mathrm{ft/s^2}$
$y_4 = A_x(t)$	longitudinal acceleration	$\mathrm{ft/s^2}$

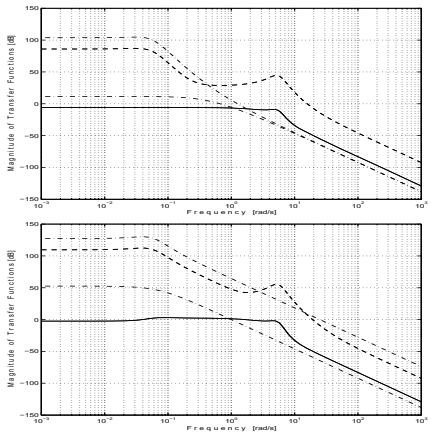


Figure 2: *a) The magnitude (maximal singular values) of transfer functions $T_{\varepsilon\omega}$ (solid line), $T_{\varepsilon m_{A_z}}$ (dash-dot lines), and $T_{\varepsilon\nu}$ (bold dashed line) for unity estimation weighting C_z and b) special weight C_z^*, respectively.*

Non-decoupling solution by means of H_∞ filtering

In the least ambitious approximation of the problem let us begin with the consideration that one cannot (or do not want to) find a robust decoupling solution. The classical solution approach, which does not necessitate to enforce any separability condition was formulated by Solution 1. According to this idea, we seek a residual generator in the form of a state observer with state $\hat{x}$ and weighted state estimate $\hat{z}$ as

$$\begin{aligned}\dot{\hat{x}} &= A\hat{x} + K\big(C(x-\hat{x})\big) + F_{A_z} m_{A_z} + B_\delta(\delta+\nu_\delta) + B_\omega\omega,\\ \hat{y} &= C\hat{x},\\ \hat{z} &= C_z\hat{x}\end{aligned} \tag{10}$$

where K is the feedback gain such that the effect of $\omega(t)$ is attenuated on the filter innovation $C(x-\hat{x})$ in sense of L_2-norm over a finite interval of time by a fixed factor γ. In effect, we want to minimize the transmission of the disturbance $\omega(t)$ by minimizing the magnification of the worst-case disturbance transfer function

$$T_{\varepsilon\kappa}(s) = C_z(sI - A + KC)^{-1}B_\kappa,$$

w.r.t. the unified fault effect characterized by

$$T_{\varepsilon f}(s) = C_z(sI - A + KC)^{-1}B_f$$

with $B_f = [B_\delta \; F_{A_z}]$ and $B_\kappa \triangleq B_\omega$. The classical optimal filtering solution of this problem (disregarding sensor noise), which makes a trade-off between worst-case disturbance κ and L_2 norm of the filter error $\tilde{z} = (C_z x - \hat{z})$ results in the classical H_∞ filtering solution, which amounts to solving the single modified algebraic Riccati equation

$$AQ + QA^T - Q\left(C^TC - \frac{1}{\gamma^2}C_z^TC_z\right)Q + B_\omega B_\omega^T = 0$$

for Q starting from a sufficiently large γ_o recursively, until $\gamma_{min} \in \mathbb{R}_+$ is found for which the constant $\gamma_{min} - \epsilon$ with an arbitrary small $\epsilon > 0$ no longer produces a positive definite solution. Then, the optimal filter gain K in (10) is obtained as QC^TC, see e.g., (Edelmayer, Bokor, & Keviczky 1994) and the references therein for details. Including sensor noise in (8), the approach would necessitate the solution of two coupled Riccati equations, see (Mangoubi 1998).

The filtering solution characterized by the respective transmission levels (fault and disturbance signal magnifications) of the filter for system (9) for unity estimation weighting C_z is shown in Fig. 2/a. By the special choice of C_z using diagonal normalization the estimation can be further improved according to Fig. 2/b. The result evidences a minimum of the disturbance transmission level at $\gamma = 1.8669$ (-5 dB correspondingly). This shows that the H_∞ filter grants at least 60 dB of separation (signal-to-noise ratio, SNR) between the sensor fault and the disturbance effect and indicates almost SNR 120 dB for the elevon fault. It can be concluded that this sensitivity is certainly enough to robustly detect both the elevon and the accelerometer faults in the practice, even in the presence of the non-decoupled disturbance. When using this solution, however, the isolation of the two fault effects using a single filter, might be problematic in the practice. In order to achieve a better fault selectivity and robustness, our design goal is to consider the solution method proposed by Solution 2 as it will be detailed in the next part.

Fault decoupling with H_∞ disturbance attenuation

Alternatively, we may attempt to separate the faults $\mu_{A_z}(t)$ and $\nu_\delta(t)$ from each other using the idea of inversion-based direct fault reconstruction. According to the idea depicted by Solution 2 this implies a two step solution procedure. In the first step of this approach we provide, if possible, a useful filtering structure for exact fault decoupling. The construction of a useful residual structure amounts finding of the inverse according to both $\mu_{A_z}(t)$ and $\nu_\delta(t)$ irrespectively of the presence of the disturbance $\omega(t)$. This will result in a residual generator of form (4)-(5) providing exact fault decoupling where the individual residual components are subject to the disturbance effect. Robust detection, therefore, necessitates the utilization of additional disturbance suppression methods. Therefore, in the second step, a disturbance attenuation method can be used to suppress the disturbance effect on the filter residuals. The design procedure can be summarized as follows.

STEP 1. (Fault decoupling residual generator design by means of system inversion). According to the idea described by Solution 2, one need to generate the system of generalized output functions containing the original measurements y and their derivatives of order r up to the input appears. Then, a selection of these output functions for the determination of distinctive input functions can be used through inversion.

Based on this consideration, it is easily seen that the measurement available for the determination of the accelerometer fault μ_{A_z} is the third output equation $y_3(t)$. Let us write, therefore,

$$y_3 = c_3^T x + \mu_{A_z} \tag{11}$$

from which the sensor fault can be expressed as the inverse function

$$\mu_{A_z} = y_3 - c_3^T x. \tag{12}$$

Portraying the fault signal ν_δ, the first derivative of the fourth measurement equation can be used because it does not necessitate the application of the derivative of the disturbance (ω does not enter this equation), letting

$$\dot{y}_4 = c_4^T \dot{x} = c_4^T Ax + c_4^T B_\omega \omega + c_4^T B_\delta(\delta + \nu_\delta). \tag{13}$$

Since, for this example, the product $c_4^T B_\omega$ is found zero, the actuator fault is obtained

$$\nu_\delta = \frac{1}{c_4^T B_\delta}\left(\dot{y}_4 - c_4^T Ax\right) - \delta. \tag{14}$$

By substituting the new output functions (12) and (14) into the state equation (9) one obtains

$$\dot{x} = Ax - \frac{1}{c_4^T B_\delta} B_\delta c_4^T Ax + \frac{1}{c_4^T B_\delta} B_\delta \dot{y}_4 + B_\omega \omega, \tag{15}$$

and, by using the definitions

$$\bar{A} = \left(I - \frac{1}{c_4^T B_\delta} B_\delta c_4^T\right) A, \qquad \text{and} \qquad \bar{B} = \frac{1}{c_4^T B_\delta},$$

one gets the representation of the inverse system from (15) by

$$\begin{aligned} \dot{x} &= \bar{A}x + \bar{B}B_\delta \dot{y}_4 + B_\omega \omega \\ \bar{y} &= \bar{C}x \end{aligned} \tag{16}$$

with

$$\bar{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad \text{and} \quad \bar{C} = \begin{bmatrix} c_1^T \\ c_2^T \end{bmatrix},$$

where the fault signals can be estimated in the form

$$f = \begin{bmatrix} \mu_{A_z} \\ \nu_\delta \end{bmatrix} = \begin{bmatrix} -c_3^T \\ \dfrac{-c_4^T A}{c_4^T B_\delta} \end{bmatrix} x + \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{1}{c_4^T B_\delta} \end{bmatrix} \begin{bmatrix} y_3 \\ \dot{y}_4 \end{bmatrix} \tag{17}$$

based on the measurement $y_3(t)$ and the first derivative of $y_4(t)$.

STEP 2. (H_∞ filter design to attenuate the effect of the disturbance on the fault decoupled residual). For disturbance attenuation both the *unknown input observer* or the H_∞ *optimal filtering* methods can be used. As the application of H_∞ filtering is potentially more flexible and more robust than other approximate detection filter design techniques, which tend to be based on geometric theory like unknown input observers, in the following we show how the H_∞ disturbance attenuation approach may contribute in finding a novel solution.

Consider the inverse system (16), with input directions $B_\Delta \triangleq \bar{B}B_\delta$ and $B_\kappa \triangleq B_\omega$, to be equivalent with the generalized representation

$$\begin{aligned} \dot{\bar{x}} &= \bar{A}\bar{x} + B_\Delta \nu_\Delta + B_\kappa \kappa \\ \bar{y} &= \bar{C}\bar{x}, \end{aligned} \tag{18}$$

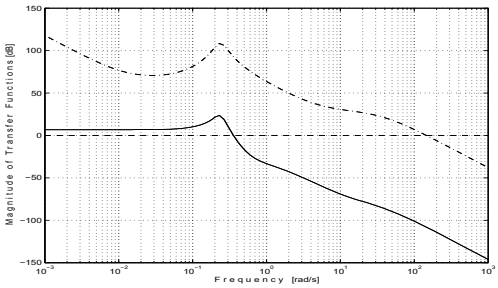


Figure 3: *Singular value plots of the combined fault to output residual transfer function $T_{\varepsilon\nu_\Delta}$ (dotted line) and wind gust disturbance to output residual transfer function $T_{\varepsilon\kappa}$ (solid line) of the detection filter derived by using standard H_∞ optimization technique with unity estimation weight C_z.*

in an attempt to design a state observer which gives an estimate $\hat{z}(t)$ of the weighted state vector

$$z = C_z x. \tag{19}$$

It can be easily checked that the observability matrix of the pair $(\bar{C}, \bar{A})$ is full rank, i.e., the system (16) and so (18) is observable. The equivalence of systems (16) and (18) can be validated by the following. On the one hand, generating (18) from (16) by substituting the inverse equations (12) and (14), respectively, we effectively combine the faults $\mu_{A_z}(t)$ and $\nu_\delta(t)$ and control input $\delta(t)$ into a new input function $\nu_\Delta(t)$ while keeping the disturbance input $\omega(t)$ separated. On the other hand, $\omega(t)$ is the only disturbance affecting the system in the predetermined direction B_ω, therefore, it can be viewed as worst-case disturbance $\kappa(t)$.

The inverse system, driven by the measurements $y(t), \dot{y}(t)$, and control input $\delta(t)$ provides the generalized inputs $\nu_\Delta(t), \bar{y}(t)$ and worst-case disturbance $\kappa(t) \triangleq \omega(t)$ for which an H_∞ filter can be designed in the form

$$\begin{aligned} \dot{\hat{x}} &= (\bar{A} - K)\hat{x} + B_\Delta \nu_\Delta + Q\bar{C}^T \bar{y} \\ \hat{z} &= C_z \hat{x}, \end{aligned} \tag{20}$$

in which Q is the solution of the corresponding modified algebraic filter Riccati equation where the filter gain is calculated as $K = Q\bar{C}^T\bar{C}$. In fact, the filter (20) provides the estimation of the dynamics of the inverse system (18). Based on the estimation $\hat{x}$, the residual (17) reconstructs the faults by means of exact decoupling in which the effect of the disturbance is suppressed in H_∞ sense.

The transmission levels of the H_∞ filter (20) designed for the generalized system (18) are given in Fig. 3. Note that the sensor fault appears in the measurements directly (see Eq. (11)) thus it has a direct 0 dB feedthrough to the residual signal. The ragged line at zero dB is the transmission of the sensor fault providing a constant level response in the whole working frequency range. Note that for the evaluation of the results this zero dB transmission is to be shifted up to the typical low frequency or steady-state transmission of the combined target fault because the optimization was done against the combined fault effect. The steady-state transmission of the combined target fault exceeds the transmission

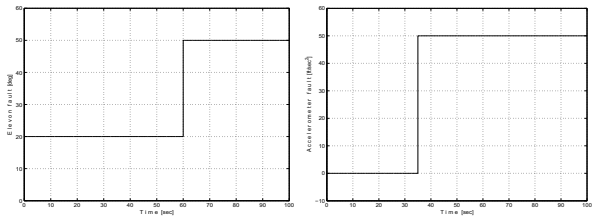


Figure 4: *Elevon and normal accelerometer faults occurring at $t = 60\ s$ and $t = 35\ s$, respectively.*

of the disturbance by more than 115 dB in the DC range and still maintains a minimum of SNR 65 dB in the frequencies over 10 rad/s. This is an excellent sensitivity for the detection of the target faults even they are continuously corrupted by the wind gust. This result is not characteristically better than those obtained by the approach presented in the previous section (and also in (Douglas & Speyer 1995; Chung & Speyer 1998)), however, it represents a significant difference. Namely, by this solution the faults are subject to exact decoupling whose effects show up in a single filter residual separately as seen in (17). Obtaining a realization of the filter (20) with the filter gain

$$K = \begin{bmatrix} 0 & 0.0015 & 0.0306 & 0 & 0.0015 \\ 0 & 0.0022 & 0.0424 & 0 & 0.0022 \\ 0 & 0.0005 & 0.0142 & 0 & 0.0005 \\ 0 & -0.0000 & -0.0029 & 0 & -0.0000 \\ 0 & 0.0048 & 0.1004 & 0 & 0.0048 \end{bmatrix},$$

a process simulation was used to evaluate the design and validate the fulfilment of the requirements. The performance of the filter designed for the detection and separation of the elevon and normal accelerometer faults in the presence of wind gust is shown in Fig. 5-6. The fault characteristic is

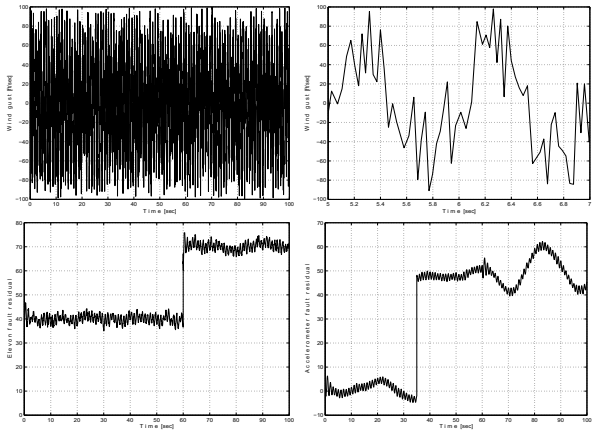


Figure 5: *Moderate energy wind gust disturbance (see the zoomed time-scale for greater resolution) and the elevon and normal accelerometer fault residuals in the presence of this moderate wind gust.*

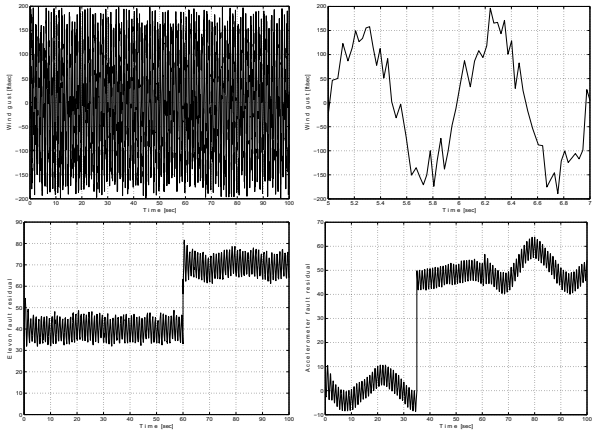


Figure 6: *Severe wind gust disturbance (see the zoomed time-scale for greater resolution) and the elevon and normal accelerometer fault residuals in the presence of severe wind gust.*

shown in Fig. 4. The values for moderate and severe wind gust composed of a smaller and a higher energy (relative to the magnitude of the anticipated faults) quasi-sinusoid signal with 1 Hz frequency and a small energy 25 Hz random signal were used for the analysis to generate the winds referred to as *severe* and *moderate* gusts in the illustrations Fig. 5 and Fig. 6, respectively.

Exact fault and disturbance decoupling

By extending the concept of Solution 2, one may attempt to separate the faults $\mu_{A_z}(t)$, $\nu_\delta(t)$ and also the disturbance $\omega(t)$ from each other providing full decoupling in the state space by direct input reconstruction. Realization of this idea in the first step, requires the calculation of the inverse not only for the fault signals $\mu_{A_z}(t)$ and $\nu_\delta(t)$ but also for the disturbance signal $\omega(t)$.

STEP 1. For the determination of the accelerometer fault μ_{A_z} the third output measurement $y_3(t)$ is available as was in case of the previous problem. Based on this measurement the sensor fault residual can be expressed as the inverse function

$$\mu_{A_z} = y_3 - c_3^T x. \tag{21}$$

As it wouldn't be a wise idea to include the derivative of the disturbance function in the residual, let the inverse calculation *w.r.t.* ν_δ, therefore, be based on the derivative of the fourth measurement equation again (ω is not included in this equation), letting

$$\dot{y}_4 = c_4^T \dot{x} = c_4^T A x + c_4^T B_\omega \omega + c_4^T B_\delta (\delta + \nu_\delta). \tag{22}$$

Since the equality $c_4^T B_\omega = 0$ holds, the actuator fault is obtained as

$$\nu_\delta = -\frac{c_4^T A}{c_4^T B_\delta} x + \frac{1}{c_4^T B_\delta} \dot{y}_4 - \delta. \tag{23}$$

In the reconstruction of the disturbance function $\omega(t)$ one may select between two opportunities: the inverse representation can either be based on $y_1(t)$ or $y_2(t)$ (and also on their derivatives). Selecting the second measurement equation $y_2(t)$ it can be written

$$\begin{aligned} y_2 &= c_2^T x \\ \dot{y}_2 &= c_2^T \dot{x} = c_2^T \big(Ax + B_\omega \omega + B_\delta(\delta + \nu)\big). \end{aligned} \tag{24}$$

By knowing the identity relation from (23)

$$\delta + \nu_\delta \triangleq \big(\dot{y}_4 - c_4^T A x\big) \frac{1}{c_4^T B_\delta} \tag{25}$$

and, by substituting (25) into (24) one obtains the disturbance residual in the form

$$\begin{aligned} \omega = -\left(\frac{c_2^T}{c_2^T B_w} - \frac{c_2^T B_\delta}{c_4^T B_\delta c_2^T B_w} c_4^T\right) Ax + \\ \frac{1}{c_2^T B_w}\dot{y}_2 - \frac{c_2^T B_\delta}{c_4^T B_\delta c_2^T B_w}\dot{y}_4. \end{aligned} \tag{26}$$

For the sake of a more compact system of notation let us introduce the identities

$$\vartheta_1 \triangleq \frac{1}{c_2^T B_\omega}, \qquad \vartheta_2 \triangleq \frac{c_2^T B_\delta}{c_4^T B_\delta}\frac{1}{c_2^T B_\omega}, \tag{27}$$

$$\vartheta_3 \triangleq \frac{c_4^T B_\omega}{c_4^T B_\delta}, \qquad \vartheta_4 \triangleq \frac{1}{c_4^T B_\delta} + \vartheta_3\,\vartheta_2. \tag{28}$$

By using (27-28) and substituting the new output functions (12), (23) and (26) into the state equation (8) one obtains the inverse dynamics

$$\dot{\bar{x}} = \bar{A}\bar{x} + \bar{B}\bar{u}, \tag{29}$$

with the system of residuals in the form calculated above

$$\begin{bmatrix} \nu_\delta \\ \omega \\ \mu_{A_z} \end{bmatrix} = \begin{bmatrix} -\dfrac{c_4^T}{c_4^T B_\delta} \\ -\dfrac{c_2^T}{c_2^T B_w} - \dfrac{c_2^T B_\delta}{c_4^T B_\delta c_2^T B_w} c_4^T \\ -c_3^T \end{bmatrix} A\bar{x} + \begin{bmatrix} -1 & 0 & 0 & \dfrac{1}{c_4^T B_\delta} \\ 0 & 0 & \dfrac{1}{c_2^T B_\delta} & -\dfrac{c_2^T B_\delta}{c_4^T B_\delta c_2^T B_w} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ y_3 \\ \dot{y}_2 \\ \dot{y}_4 \end{bmatrix}, \tag{30}$$

where

$$\begin{aligned} \bar{A} &= \Big(I + B_\omega\big(\vartheta_2 c_4^T - \vartheta_1 c_2^T\big) + B_\delta\big(\vartheta_3\vartheta_1 c_2^T - \vartheta_4 c_4^T\big)\Big)A, \\ \bar{B} &= [\,0 \quad 0 \quad 0 \quad (\vartheta_1 B_\omega - \vartheta_3\vartheta_1 B_\delta \quad -\vartheta_2 B_\omega + \vartheta_4 B_\delta)] \\ \bar{u} &= [\delta \quad y_3 \quad \dot{y}_2 \quad \dot{y}_4]^T. \end{aligned}$$

Since $c_4^T B_\omega$ is obtained zero the identities (27-28) and the matrices $\bar{A}, \bar{B}$ reduce to

$$\vartheta_1 \triangleq \frac{1}{c_2^T B_\omega},\ \vartheta_2 \triangleq \frac{c_2^T B_\delta}{c_4^T B_\delta}\frac{1}{c_2^T B_\omega},\ \vartheta_3 = 0,\ \vartheta_4 \triangleq \frac{1}{c_4^T B_\delta},$$

$$\bar{A} = \Big(I + B_\omega\vartheta_2 c_4^T - B_\omega\vartheta_1 c_2^T - B_\delta\vartheta_4 c_4^T\Big)A,$$

$$\bar{B} = [\,0 \quad 0 \quad 0 \quad \vartheta_1 B_\omega \quad -\vartheta_2 B_\omega + \vartheta_4 B_\delta\,].$$

Rendering the estimate $\hat{\bar{x}}(t)$ available, the calculation of the residuals $\nu_\delta(t), \mu_{A_z}(t)$ and $\omega(t)$ from (30) can be made.

STEP 2. (State observer design to estimate the state of the inverse dynamics). Relying on Proposition 1 exerted by Solution 2, formulate the observer design problem for estimating the unknown state of the inverse dynamics based on system (29) and the auxiliary measurement equations

$$\begin{aligned} \dot{\bar{x}} &= \bar{A}\bar{x} + \bar{B}\bar{u}, \\ \bar{y} &= \bar{C}_A\bar{x} \end{aligned} \tag{31}$$

where $\bar{y} \triangleq y_1$ and $\bar{C}_A \triangleq c_1^T$ by definition. Observe that y_1 is the only measurement that were not utilized in the determination of the residuals represented by (12), (23) and (26), respectively.

It can be easily checked that the pair $(\bar{C}_A, \bar{A}) = (c_1^T, \bar{A})$ is non-observable. By appending the original measurements to $\bar{y}$ (all but not y_3), by constructing the output matrix $\bar{C}_A = [\,c_1^T \;\; c_2^T \;\; c_4^T\,]^T$, however, the auxiliary system $(\bar{C}_A, \bar{A})$ can be made observable that makes the observer design problem viable.

Based on the representation $(\bar{C}_A, \bar{A})$ let us construct a Luenberger state observer to obtain an estimate $\hat{\bar{x}}$ of the inverse state $\bar{x}$ in (31). The problem of finding a feedback matrix K, such that the closed loop observer gain $\bar{A} + K\bar{C}_A$ is stable and the filter satisfy some performance requirements (*i.e.*, it has the desired set of eigenvalues) is a standard problem of modern control theory which has a number of solution methods available.

By posing the design requirement that the filter eigenvalues along the real axis are smaller than -0.5, a stable filter with an acceptable transient behavior can be obtained with

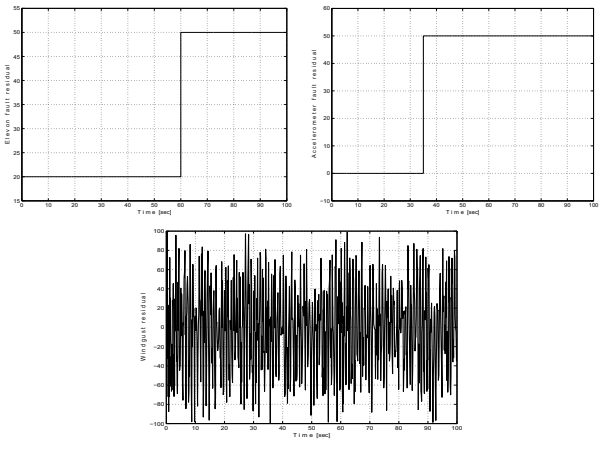


Figure 7: *The separated filter residual components driven by the elevon fault ν_δ, normal accelerometer fault μ_{A_z}, and windgust disturbance ω, respectively.*

using the eigenvalue assignment approach yielding to observer gain K with stable spectrum as

$$K = \begin{bmatrix} 20840 & -162670 & -70140 \\ 32730 & -255510 & -110200 \\ 30 & 20 & 10 \\ -2670 & 20790 & 8970 \\ -32730 & 255490 & 110210 \end{bmatrix}. \quad (32)$$

The result of a continuous time process simulation assuming the simultaneous occurrence of an elevon fault and a normal accelerometer fault (as shown in Fig. 4) in the presence of persistent wind gust disturbance is presented in Fig. 7. The result evidences that, in this noise free case, the filter gives detection and estimation of the two fault signals ν_δ, μ_{A_z} *separately*, disregarding the disturbance effect ω in the residual. It shows that the disturbance ω, and the faults ν_δ, μ_{A_z} show up in three independent residual directions making the robust detection and separation of the fault signals, with good sensitivity measure, possible.

CONCLUSIONS

In this paper the effectiveness of inversion-based direct input reconstruction for robust detection and separation of multiple simultaneous faults in the presence of persistent disturbances which proved to be non-separable in traditional ways in linear dynamical systems is presented. It is shown how in a specific filtering structure, relying on the inverse representation of the system, a detection filter can be designed by means of the estimation of the state of the inverse dynamics. One can use standard state observers or H_∞ filters combined with the inverse-based residual generator for providing full or approximate decoupling solutions, respectively.

Applicability of the method requires more measurements than faults available, as well as the access to derivative measurements of some signals. The number and structure of available measurement data is an issue of system specification and design. In light of the development of recent sensor technology, the need for direct access to derivatives is a technical reality, however, noise sensitivity of the method is a further issue of research. In this paper, this novel fault detection and isolation method was presented for noise free case. Further studies to ensure robustness of the theory against measurement noise and system modeling errors are needed.

References

Beard, R. V. 1971. *Failure accomodation in linear systems through self-reorganization*. Ph.D. Dissertation, Department of Aeronautics and Astronautics, MIT, Cambridge, MA. MVT-71-1.

Chen, J., and Patton, R. J. 2000. Standard H_∞ filtering formulation of robust fault detection. In Edelmayer, A., ed., *Proc. 4th IFAC Symp. on Fault Detection, Supervision and Safety for Technical Processes, Safeprocess'00*, 261–266. ISBN 0 08 043250 6, Elsevier Science Ltd, Oxford, UK.

Chung, W. H., and Speyer, J. L. 1998. A game theoretic fault detection filter. *IEEE Trans. Aut. Cont.* 43(2):143–161.

Douglas, R. K., and Speyer, J. L. 1995. An H-infinity bounded fault detection filter. In *Proc. Amer. Cont. Conf.*, 86–90. Seattle, WA.

Douglas, R. K., and Speyer, J. L. 1999. H_∞ bounded fault detection filter. *J. Guidance, Cont. and Dynamics* 22(1):129–138.

Edelmayer, A., and Bokor, J. 2002. Optimal H_∞ scaling for sensitivity optimization of detection filters. *Int. J. of Nonlin. and Robust Cont.* 12(8):749–760. John Wiley, London.

Edelmayer, A.; Bokor, J.; Szabó, Z.; and Szigeti, F. 2004. Input reconstruction by means of system inversion: a geometric approach to fault detection and isolation in nonlinear systems. *Int. J. Appl. Math. Comp. Sci.* 14(2):101–111.

Edelmayer, A.; Bokor, J.; and Keviczky, L. 1994. An H_∞ filtering approach to robust detection of failures in dynamical systems. In *Proc. 33rd IEEE Conf. Dec. Cont*, 3037–3039. Orlando, FL.

Edelmayer, A.; Bokor, J.; and Keviczky, L. 1996. H_∞ detection filter design for linear systems: Comparison of two approaches. In *Proc. of the 13th IFAC World Congress*, 37–42. San-Francisco, CA.

Edelmayer, A.; Bokor, J.; and Szabó, Z. 2006. Robust detection and estimation of faults by exact fault decoupling and H_∞ disturbance attenuation in linear dynamical systems. In *Proc. Amer. Cont. Conf., ACC'06*, 5716–5721. Minneapolis, MN.

Edelmayer, A.; Vera, C. E.; and Szigeti, F. 2002. A geometric view on inversion based FDI in linear systems. In *CD-ROM Proc. Mediterranean Conf. Cont. Aut, MED'02*. Lisbon, Portugal.

Frank, P. M., and Ding, X. 1994. Frequency domain approach to optimally robust residual generation and evaluation for model-based fault diagnosis. *Automatica* 30(5):789–804.

Isidori, A. 1985. *Nonlinear Control Systems*. Springer-Verlag, Berlin. (Third Edition 1995).

Mangoubi, R. S. 1998. *Robust Estimation and Failure Detection – A Concise Treatment*. Springer-Verlag, London.

Nijmeijer, H., and Van der Schaft, A. J. 1991. *Nonlinear Dynamical Control Systems*. Springer-Verlag, London.

Szigeti, F.; Bokor, J.; and Edelmayer, A. 2002. Input reconstruction by means of system inversion: application to fault detection and isolation. In *CD-ROM Proc. 15th IFAC World Congress*. Barcelona, Spain.

White, J. E., and Speyer, J. L. 1987. Detection filter design: Spectral theory and algorithms. *IEEE Trans. Aut. Cont.* AC-32(7):593–603.

Obtaining Models for Test Generation from Natural-language-like Functional Specifications

M. W. Esser[1], P. Struss[1,2]

[1]Technische Universität München, Boltzmannstr. 3 D-8578 Garching, Germany
[2]OCC'M Software, Gleissentalsstr. 22, D-82041 Deisenhofen, Germany
{esser, struss}@in.tum.de, struss@occm.de

Abstract

The paper presents first results of a project that aims at a model-based tool for functional testing of control software for passenger vehicles. The objective is that this tool can be used in today's engineering practice and, hence, the approach must not require costly changes in the current test generation process and not assume data and skills that do not exist in reality. Firstly, the input to test generation, i.e. the specification of functional requirements, exists mainly as natural language text, rather than in a formal language. Secondly, its output, i.e. a set of proposed tests, has to be justified and explained in terms that are comprehensible to the test engineers in order to allow them to inspect, revise, and complement it. We focus on design decisions that are motivated by this objective. The proposed solution offers a natural-language-template-based interface for acquiring software requirements. The content of the filled-in templates can be represented in propositional logic and temporal relations and form the model of the intended correct behavior. Models of potential faulty behaviors are generated from this OK model by a number of (types of) transformations. The fault types are defined mainly to match the intuition behind manually generated test cases and, hence, can deliver similar, but more systematic, test suites. This forms the basis for the intuitive justification of the tests and its manual post-processing.

1. Introduction

As long as formal specification of software functions and software generation and verification based on formal specifications are not established, testing of software is an essential step in software development. And even if, some day, the precondition would be achieved, there would be no guarantee that the starting point, the specification, captures the intuitive user expectations appropriately, which again establishes a need for testing. This holds even more if the software has a high impact on safety of people and/or environment such as the control software on vehicles.

Automotive companies spend high efforts on testing software, often delivered by suppliers, on test benches with "hardware in the loop" [Boot et al. 99] describes automated testing techniques for HIL). Figure 1 shows an exemplary test bench. Large parts of the physical behavior of the vehicle are simulated, and the electronic control unit (ECU) is tested in this context. Today, the test suites for this process are generated by hand, based on the information contained in the requirement documents. This results in high development costs and does not provide a reliable assurance of the scope of testing.

In a project with a major German car manufacturer, we aim at addressing these issues by producing a tool that generates sets of tests automatically from the requirement specification. Since the software is to be tested in the context of the (simulated or real) vehicle, a coherent approach to testing software and physical parts in a uniform way is attractive. We are addressing this requirement by extending our model-based test generation algorithm [Struss 94] to software testing. A feasibility study for this approach was presented in [Esser-Struss 07].

The major challenge of the project is to build a tool that supports the existing work process, rather than requiring a major revision of the process, long education of the involved staff, etc. This implies, in particular,

- on the input side: we cannot expect the requirement engineers to learn and use a formal language for formulating requirements. The current requirement documents state mainly **functional** requirements and are mainly natural language texts.

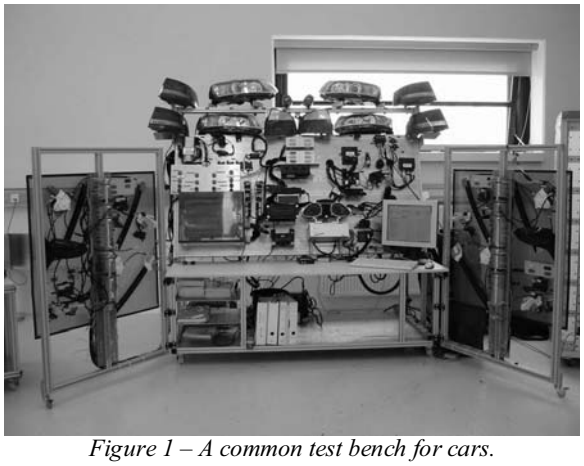


Figure 1 – A common test bench for cars.

- on the output side: the automatically generated tests need to be inspected, revised, and extended by the test engineers and, therefore, presented, justified, and explained in intuitive terms.

In this paper, we focus on the attempt to address these requirements. The proposed solution offers natural-language-template-based interface for acquiring software requirements (section 3). The content of the filled-in templates is automatically transformed into a representation in propositional logic and temporal relations and forms the model of the intended correct behavior (section 4). Models of potential faulty behaviors are generated from this OK model by a number of (types of) transformations (section 5). The fault types are defined mainly to match the intuition behind manually generated test cases and, hence, can deliver similar, but more systematic, test suites. This forms the basis for the intuitive justification of tests and their postprocessing (section 6).

In the following section, we give an overview of the approach and its elements.

2. The Approach

The perspective on testing is that confirmation of one behavior mode (OK) requires discriminating it from all possible faults. In the model-based test generation algorithm presented in [Struss 94], models (of physical systems) are represented as (finite) relations. Useful test inputs (the vertical axis in figure 2) are computed as the complement of those that may trigger the same observable response under both behavior modes (the horizontal axis). Such a test *guarantees* to refute at least one of the two beheviors.

The task of test generating for conformance testing is finding a set of test cases such that for each pair (m_{ok}, $m_{fail,i}$), where m_{ok} is the model of the correct system and $m_{fail,i}$ is one out of a given set $\{m_{fail,1}, \ldots, m_{fail,n}\}$ of models of system faults, a definitely discriminating test exists in the set.

Applying this approach to test generation of software requires a process containing at least three steps (see figure 3):

1st Step: building a model of correct behaviour, m_{ok}, In our case, the OK behavior is (solely) given by the (high-level, functional) requirements,

2nd Step: deriving fault models $\{m_{fail,1}, \ldots, m_{fail,n}\}$ from m_{ok} according to some selected fault classes,

3rd Step: computing the test cases for m_{ok} and $\{m_{fail,1}, \ldots, m_{fail,n}\}$.

The framework presented here can be seen as a refinement of these three steps under our objective, namely supporting the existing work process. Actually, it involves two different types of experts (and, in fact, different departments): requirement engineers and test engineers. The latter needs the results of the former as an input to his work and this exchange happens via documents, discussions, and phone calls and ist time-consuming, error prone, and only weakly supported by software. Our tool can be understood as providing support to both types of experts and a channel for the exchange of well-defined information between them.

In order to develop the foundation for a solution, we participated in both work processes, requirement acquisition and test generation, related to a new version of the Automatic Cruise Control System (ACC). This enabled us to develop the current working hypotheses for an appropriate representation of requirements and for identifying fault models that corresponds to the potential misbehaviors that the engineers hypothesize and their tests check for. This requires two more steps, namely

0th Step: obtaining a formal requirement specification from experts who are accustomed to using natural language for this purpose

4 th Step: presenting the generated tests and explaining the rationale behind them to the test engineers.

Figure 3 also indicates the specific requirements in the process which are:

A. The acquisition of requirements from the requirement engineer must be in a form that is close to their intuitions and the language they use, which is natural language.
B. The model (for m_{ok}) must be a formal one and
C. must preserve the structure of the requirement specification.
D. The model formalism used for the fault models must be expressive enough to cover all relevant requirements and fault types.
E. The purpose of the generated test cases must be comprehensible to the human expert.

Preserving of the structure means that the individual elements of the (manually created) requirement specification can still be identified in the model m_{ok}. The reasons for this, which are explained in detail later, are:

- to apply the same test criteria as in the current manual practice
- to explain the reason for a test case in terms of the requirement specification, which leads to intuitive justifications.

These aims are addressed in this framework by introducing two representations used in different steps in the process:

1. At the user interface: **Template Based Natural Language Specification** (TBNLS), where each requirement is a filled template forming a natural language sentence.

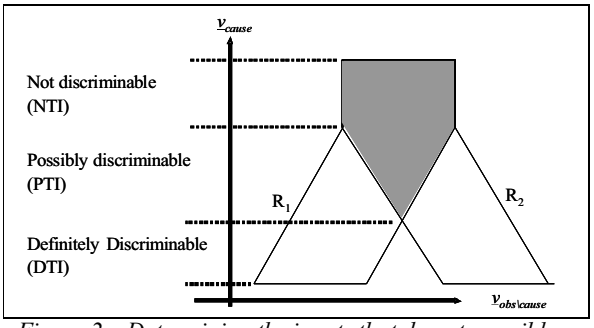


Figure 2 – Determining the inputs that do not, possibly and definitely discriminate between models R_1 and R_2.

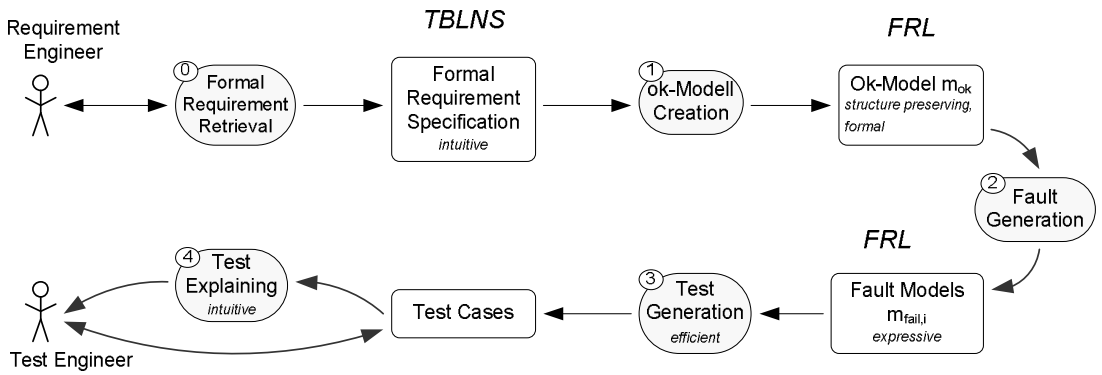


Figure 3 – the steps of fault-based test generation, and the formalisms used in this framework.

2. Internally, this is transformed into a **Formal Requirement Language** (FRL) where requirements are tuples (start-condition, consequence, end-condition). Conditions and consequence are specified in terms of propositions, that characterize system states, time intervals, and temporal constraints.

The human expert interacts only with TBNLS notation and only specifies the correct behavior. From this, the FRL specification is automatically generated and then used to generate fault models (see figure 3).
In the following sections the formalisms are described.

3. Natural-language-like Specification

In current practice, functional requirements (except for a small set of critical applications) are usually stated informally in natural language. A requirement acquisition tool has to allow the expert to stick to this as far as possible. Therefore, we decided to use a natural-language-like specification. However, since it must provide the basis for the following formal representations and algorithms, it has to

- have precise semantics and
- cover at least the most important/common classes of the requirements.

In order to achieve this, we studied current practice requirement documents supplied by the industrial partner. The left-hand column of figure 4 lists three typical examples from such a document. Our analysis showed that almost all requirements can be structured using three elements: *start-condition (if)*, *consequence (then)* and *end-condition (until)*. If a situation matches the start-condition, then also the consequence has to hold. Additionally, the termination of the consequence is specified by an *end-condition*. Thus, e.g. requirement R_2 can be reformulated as follows:

if the system is in mode m_1, lamp L3 is off, and button B4 is released,
then immediately lamp L3 is lit
until button B4 is down again or the system leaves m_1.

The end-condition may be missing, and, hence, the duration of the consequence is unspecified. If the start condition is also omitted, the consequence has to hold universally.
The following description of the template-based natural-language-like specification (TBNLS) comprises

- the set of templates for representing requirements, and
- a domain theory containing additional background information necessary for situating the requirements in a context and for generating tests that reflect and explaining this context.

Sentence Templates for Requirement Rules

A sentence template is a particular grammatical natural language pattern, which can be filled with state expressions and metrical time information. The right-hand column in figure 4 shows the filled in templates for the three sample requirements.
The template itself fixes the *temporal* relationships (e.g. P_1 must occur *before* P_2) between situations characterized by state expressions and classifies the state expressions as start-condition, consequence and end-conditions. The ability to specify exact temporal relationships is missing in many other natural language representations, such as ACE (Attempt to Controlled English, [Fuchs-Schwitter 96]) which is a subset of English restricted in vocabulary and grammar.
A state expression P characterizes a class of situations and is inductively defined from facts $F \in FACTS$:

$P := F \mid (P_1 \text{ AND } P_2) \mid (P_1 \text{ OR } P_2) \mid \text{NOT } P_1$

Facts are atomic propositions and do not have a structure or explicit semantics a priori; a reader must know what they mean. However, dependencies among facts may be defined in the domain theory. E.g. in requirement R_1 the facts are 'Button B_4 is not down', 'Button B_4 is down' and 'Lamp L_3 is lit'. '5 seconds' is metrical time information.
The choice to use unstructured propositions was made in order to avoid putting too much burden on the requirement engineer by forcing him to introduce certain predicates, variables with associated domains etc. While it is straightforward to assign a value '30km/h' to the quantity 'velocity', in other cases, it would be unnatural for requirements like 'Transition from mode m1 to m2 must be comfort-

able'. Later, we plan to explore the gain of a more structured representation of the facts. A more complex representation (which means more *work!*) may be more acceptable if the resulting benefit can be demonstrated and quantified.

In a template, 'A occurs' always means that A was false before and changed to true now, and 'A holds' means, that A is true independent of its value before or after.

The user can choose to activate a default rule, stating that a fact persists, unless specified otherwise in the template (which addresses the frame problem). However, this leads to disadvantages compared to specifying the behavior for entire time intervals explicitly: leaving the behavior unspecified for certain intervals, which may be adequate during design stage, is not possible.

Domain Theory

The functional requirements are considered as a description of the intended behavior of the respective subsystem. However, it is only a partial description and is likely to be insufficient for generating reasonable tests for several reasons. Here, we are not referring to the fact that the requirement engineer may have forgotten to specify some relevant aspects of the desired system behavior. The sources of incompleteness are more fundamental:

- The collection of requirements may **not** contain the most **basic** ones, because they are obvious to anybody involved in the process. (Actually, sometimes, they will only be assumed to be obvious, and e.g. the test engineer may not be aware of them, which creates a problem). For instance, the fundamental function of a brake, namely to decelerate the vehicle will probably not be subject to an explicit requirement.
- The collection of requirements concerns only **one subsystem**, and, more specifically, its software and does not specify the behavior of the context, i.e. the physical components of this subsystem and other subsystems it is interacting with. For instance, the Automatic Cruise Control (ACC) interacts with the braking system, whose function will produce in turn an impact on the ACC (by reducing the speed and potentially increasing the distance to the preceding vehicle).
- Also, the **environment** of the **vehicle** (road conditions, the driver's actions, other vehicles etc.) will usually not show up in the requirements, but may be relevant to test generation. For instance, acceleration of the preceding vehicle influences the distance to it, which is a measured input to the ACC. Also, basic physical constraints will be missing, such as the fact that the vehicle cannot brake and accelerate at the same time.

If this background knowledge is not available to the test generation algorithm, it may still be able to produce tests, but they may be unintuitive and overly complicated, and miss some "obvious" solutions. Complementing the model obtained from the requirements by this kind of knowledge will make automated test generation more powerful and provide results with higher acceptance. We expect that a very basic and qualitative model of vehicle functions and

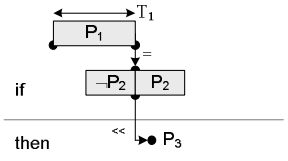
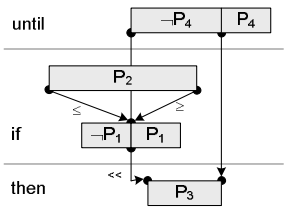
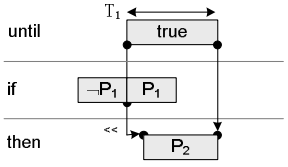
	Prosa Requirement	Requirement Sentence Template	In FRL $R=\left(\overset{\text{then}}{A}\rightarrow\overset{\text{until}}{B}\underset{\text{first}}{\rightarrow}C\right)$	Graphical
R_1	If button B4 has not been down during the last 5 seconds, then lamp L3 must be lit immediately after button B4 changed to down.	If Button B_4 is not down P_1 held the last 5 seconds T_1 and now Button B_4 is down P_2 occurs, then must Lamp L_3 is lit P_3 follow immediately.	$A=[P_1]_{t1}^{t2}\wedge[[P_2]_{t3}^{t4}\wedge(t_2\mid t_3)\wedge(t_1=t_2-T_1)$ $B=[P_3]_{t5}^{t6}\wedge(t_3\mid t_5)$	
R_2	Immediately after button B4 Is released while the system is in mode m_1 and lamp L3 is not lit, the lamp L3 must be lit until button B4 is down again or the system leaves m_1.	If Button B_4 is down P_1 occurs, during System is in mode m_1 AND Lamp L_3 is not lit P_2 holds, then Lamp L_3 is lit P_3 must hold immediately, until Button B_4 is up OR System is not in mode m_1 P_4 hold.	$A=[[P_1]_{t_1}^{t_2}\wedge[P_2]_{t_3}^{t_4}\wedge(t_3<t_1<t_4)$ $B=[P_3]_{t_5}^{t_6}\wedge(t_1\mid t_5)$ $C=[P_4]_{t_7}^{t_8}\wedge(t_6\mid t_7)$	
R_3	Immediately after button B4 Is pressed, Lamp L3 must be red for 10 seconds.	If Button B_4 is down P_1 occurs, then Lamp L_3 is red P_2 hold immediately, until 10 seconds T_1 elapsed.	$A=[[P_1]_{t_1}^{t_2}$ $B=[P_2]_{t_3}^{t_4}\wedge(t_2\mid t_3)$ $C=(t_4=t_3+10s)$	

Figure 4 – Examples of natural-language requirements and the respective templates, their notation in FRL, and graphical representation (from left).

its context will suffice to achieve this and, therefore, be highly reusable for different subsystems (and, in fact, for different work processes).
There is another limitation that is due to the chosen granularity of the current requirement representation:

- Since the entries in the templates are treated as elementary propositions, their ontological relations (e.g. exclusiveness or a taxonomy) are not made explicit in the set of requirements. In our example, 'Button is down' is the negation of 'Button is up'.

To exclude situations which are impossible in reality, such kind of dependencies have to be included in the domain theory. Otherwise, test generation, may produce test cases that cannot be executed.
The axioms of the domain theory should be expressed in the same way as the requirements (e.g. for defining that the 'Button is pushed' holds, if and only if 'Button is down' holds now and 'Button is up" was true before), but represented separately from them, because

- the requirement engineer is only interested in the requirements and should not be forced to deal with the domain theory (which is obvious background knowledge to him)
- the domain theory plays a different role in the process, in that its interdependencies are not subject to testing,.

At present, we use 15 different requirement sentence templates. They suffice to express the set of requirements in the current project. Of course, additional templates may be added if needed. Because it might be difficult to pick the right template from a larger set and because there may be a need to modify the template while formulating a requirement, in the future, we may consider supporting the construction of templates from elementary fragments (logical connectors, temporal constraints) similar to configuring functions in Excel. In this paper, we refer only to the templates underlying R_1, R_2 and R_3.

4. Formal Requirement Language

In order to build the model, the first step after acquiring the requirements in template form is to transform each template instance into a requirement in the formal language (FRL). Figure 4 shows the example rules in FRL notation (left-hand column) and their graphical illustrations.
The set of FRL requirement rules over a set of facts FACTS is defined inductively.

Definition – FRL Requirement
Let ε_{state}=P be the set of state expressions defined in section 3 and ~ be one of the relations =, <, >, ≤, ≥, and |. Then

$$\varepsilon_{interval} = \left[\varepsilon_{state}\right]_{t1}^{t'2} \mid \left[\left[\varepsilon_{state}\right]_{t1}^{t'2} \mid \left[\varepsilon_{state}\right]\right]_{t1}^{t'2} \mid \varepsilon_{interval} \wedge \varepsilon_{interval} \mid \varepsilon_{interval} \vee \varepsilon_{interval} \mid (\varepsilon_{interval}) \mid \varepsilon_{interval} \wedge \varepsilon_{quant} \mid \neg\varepsilon_{interval}$$

$$\varepsilon_{quant} = \exists\, \varepsilon_{unquant} \mid \neg\exists\, \varepsilon_{unquant} \mid \varepsilon_{quant} \wedge \varepsilon_{quant} \mid \varepsilon_{quant} \vee \varepsilon_{quant}$$

$$\varepsilon_{unquant} = (\varepsilon_{interval} \wedge \varepsilon_{constr}) \mid \neg\varepsilon_{unquant}$$

$$\varepsilon_{constr} = (t_i \sim t_j) \mid (t_i \sim t_j + x) \mid (t_i \sim t_j - x) \mid \varepsilon_{constr} \wedge \varepsilon_{constr} \mid \varepsilon_{constr} \vee \varepsilon_{constr}$$

$$\varepsilon_{requ} = \left(A \xrightarrow{\text{then}} \left[\varepsilon_{state}\right]_{ta}^{tb} \wedge \varepsilon_{constr} \xrightarrow[\text{first}]{\text{until}} C \right) \mid \left(A \xrightarrow{\text{then}} \left[\varepsilon_{state}\right]_{ta}^{tb} \wedge \varepsilon_{constr} \right)$$

where A,C ∈ $\varepsilon_{unquant}$.
The informal semantics is:

- $\left[\varepsilon_{state}\right]_{t1}^{t'2}$ specifies that ε_{state} holds all the time during the (closed) interval $[t_1,t_2]$, where ε_{state} may but not need to hold before and after that interval.
- $\left[\left[\varepsilon_{state}\right]_{t1}^{t'2}$ specifies a left-max interval where ε_{state} must hold, i.e. additional to the above expression, ε_{state} must not hold in the interval right before t_1. The analogue holds for $\left[\varepsilon_{state}\right]]_{t1}^{t'2}$.
- $t_1 \mid t_2$ means t_2 follows t_1, i.e. there is an infinitely short time between t_1 and t_2.

The formal semantics is given in predicate logic, where F(t) means, that fact F holds at time t:

$$\left[\varepsilon_{state}\right]_{t1}^{t'2} := \forall_{t,t1\le t\le t2} \varepsilon_{state}(t)$$

$$\left[\left[\varepsilon_{state}\right]_{t1}^{t'2} := \exists_{t',t0} \left[\neg\varepsilon_{state}\right]_{t'}^{t0} \wedge \left[\varepsilon_{state}\right]_{t1}^{t'2} \wedge (t_0 \mid t_1)\right.$$

$$t_1 \mid t_2 := \forall_{t0<t1,t3>t2} \left([t_0;t_1] \cup [t_2;t_3] = [t_0;t_3] \right)$$

where [a;b] is the closed interval between a and b. Figure 5 shows the semantics of a requirement in predicate logic. $\overline{C}$ is a copy of C, where each variable name v in v(C), except those occurring also in formula A, is renamed into $\overline{v}$. The same holds for $\overline{B}$.

Preservation of Structural Information

Requirements are constraints on the intended behavior of the system. While in verification one would simply check whether or not the set of constraints are fulfilled, test generation and test justification (section 6) requires the preservation of some structural information. From a purely logical perspective, one could transform an FRL requirement into an implication and, thus, into a single constraint and a set of requirements into a constraint network. While this can be done to specify the semantics of $\xrightarrow{\text{then}}$ and $\xrightarrow[\text{first}]{\text{until}}$,
FRL maintains

- the individual requirements as units and
- within a single requirement, the distinction between start-condition, consequence and end-condition.

This is essential because the tests have to be generated for each single requirement, and conditions and consequence play a different role in test generation. While the former (and their mutations) need to be **established** by other **actions**, the latter is what must be established by the system, can be affected by faults, and, hence, needs to be **checked**. Section 5 will show how this structure is exploited for generating fault models.

$$R = \left(A \xrightarrow{\text{then}} B \xrightarrow[\text{first}]{\text{until}} C \right) := \forall_{\text{fv}(A)} \left(A \Rightarrow BC_1 \vee BC_2 \right),$$

$$R = \left(A \xrightarrow{\text{then}} B \right) := \forall_{\text{fv}(A)} \left(A \Rightarrow C \right)$$

with

$$BC_1 \Leftrightarrow \exists_{ta,tb} \left(B \wedge \left(\exists_{\text{fv}(C)} C \right) \wedge \left(\neg \exists_{\text{v}(\overline{C}),\overline{ta},\overline{tb}<tb} \left(\overline{C} \wedge c_{\overline{B}} \right) \right) \right)$$

$$BC_2 \Leftrightarrow \left(\exists_{t_a,t_b} \left(B \wedge \left(t_b = \infty \right) \right) \wedge \forall_{t_a,t_b} \neg \exists_{fv(C)} \left(C \wedge c_B \right) \right)$$

$$B := \left[P \right]_{t_a}^{t_b} \wedge c_B$$

Figure 5 – Semantic of a FRL requirement rule in predicate logic.

Other possible representations, such as finite state machines representing the whole functionality of the system or formulae of the Duration Calculus (DC, see [Chaochen-Hansen 03]) do not contain information about start-, end-condition or consequence of the original requirements.
Requirements in FRL do not only contain this structural information, but also provide a 1:1 mapping between each state expression in the template and in the formal requirement. This is relevant for the tool, because it forms the basis for presenting comprehensible justifications for the generated tests to the test engineer, which can be stated in terms of the elements of the original requirements (formulated in a template).

Expressiveness of FRL

The formal requirement language is quite expressive, and, in fact, it is overly expressive from the application point of view. The included continuous time model results in undecidability. For instance, consider the simple rule: '5 seconds after pressing a button, the lamp must be lit'. The button could be pressed infinitely often during 5 seconds, and each time the requirement would have to hold. Thus during this period, an infinite amount of memory would be necessary to keep track of the individual time points the lamp must be lit. Of course, this is irrelevant under practical consideration, because the button may be pushed often, but not infinitely often.
There are two ways to cope with this issue. One could restrict the language explicitly to a discrete time model with finite granularity. Alternatively, one can restrict the use of the language. This is what is currently guaranteed by the finite sets of templates (and requirements), the way fault models are constructed, and the fact that test generation considers a finite set of steps only.

5. Fault Model Types

We also analyzed how test cases of an existing test suite were generated and identified the motivation behind them. It is not surprising that tests are explicitly or (often) implicitly based on hypothesizing "what may go wrong" and designed to detect a certain type of misbehavior (deviation from the specification). It turned out that many of these fault types can be represented as defects in the requirement specification and can be described in terms of start-condition, end-condition and consequence that are "mutations" of the ones occurring in the requirements.
In the following, we discuss some fault types that were used most frequently in the analyzed documents. We present examples and the fault types informally and in FRL.
Two of the most obvious fault types are, stated intuitively,

- The conditions are satisfied, but the consequence does not occur.
- The condition is not satisfied, but the consequence occurs, anyway. (At a second glance, this is not necessarily a fault, unless there are other requirements contradicting this).

To illustrate the first case, consider the example requirement R: "if button B3 is down, lamp L1 is lit and the speed is below 130km/h, lamp L2 must be on for 5 seconds".
Then the first fault type states that in any situation matching the start-condition (*positive* case), the lamp L2 is not on for 5 seconds (but may be on for less than 5 seconds).

Simple Positive Fault

Contrary to the specification, in all conditions that satisfy the start condition of a requirement, its consequence does not occur, i.e. the negated consequence occurs

For a requirement R_i one fault model $m_{\text{fail},i}$ is created which differs from the correct specification m_{ok} only in R_i by replacing B with its negation:

$$m_{\text{fail},i} = m_{\text{ok}} \left[R_i \mapsto R_{i,\text{fail}} \right] \wedge$$

$$R_{i,\text{fail}} = R_i \left[\left(B \xrightarrow{\text{until}} C \right) \mapsto \neg \left(B \xrightarrow{\text{until}} C \right) \right].$$

where $X[Y \mapsto Z]$ means that Y is replaced by Z within X.

Here, and in the following, the respective requirements are stated as

$$R_i = \left(A \xrightarrow{\text{then}} B \xrightarrow[\text{first}]{\text{until}} C \right)$$

For the next fault type assume that it is known (via other requirements) that if even only one of the conjuncts in the start-condition of R is not true (*negative* case), e.g. the button is not down but L1 is off and speed is below 130km/h, then the lamp L2 is not on for 5 seconds. In contrast the fault states that L2 *is lit* in such negative cases.

Simple Negative Fault

Contrary to the specification, in all conditions that differ from the start-condition of a requirement by exactly one negated conjunct, the requirements consequence does occur, although other requirements R_2, … R_n implies a different consequence.

For each fact occurrence FO_j in start-condition A of a requirement R_i, a fault model $m_{\text{fail},i,j}$ is created which differs from the ok model by removing FO_j from A:

$$m_{\text{fail},i,j} = m_{\text{ok}}\left[R_i \mapsto R_{i,\text{fail},j}\right] \wedge$$
$$R_{i,\text{fail},j} = R_i\left[A \mapsto A_{\text{fail},j}\right] \wedge$$
$$A_{\text{fail},j} = A\left[FO_j \mapsto \text{true}\right].$$

The next fault type leads to a boundary value analysis. The fault states that in any borderline situation, like the speed is 129,5km/h, the consequence does not occur.

We therefore define that a value assignment v='(v_1,…v_n) for all variables V=(V_1,…V_n) is called a positive borderline situation of an expression expr, iff the expression is true under v and there exists a i≤n such that expr becomes false when in- or decreasing (only) the value v_i.

Boundary Value Positive Fault

Contrary to the specification, for each positive borderline situation of the start-condition, the consequence does not occur.

Analogously, a negative borderline can be defined, where the expression is false but becomes true with an in-/decrease, leading to the definition of a Boundary Value Negative Test.

Representing this fault type is fairly clumsy using propositions only. It will be easier when we introduce value assignments to variables as state expressions..

Although the simple positive fault type appears very simple, there can be many state expressions that satisfy the respective condition (conjunctions that subsume the condition), and only some of them are reasonable to consider, because additional conjuncts interact with the other ones. This interaction can be due to shared resources. E.g. assume that both the lamp L1 and the window lifter consume a significant amount of power from the battery. A faulty behavior because of resource shortage may occur if both, 'L1 is lit' and 'window lifter is on' hold at the same time.

Resource Shortage Fault

The starting conditions of two requirements are satisfied such that both consequences must hold during the same time, where both shares a common resource, one of the consequences does not occur.

For each tuple (F_1,F_2)∈FACTS^2 where both facts share the same resource, *share-resource(F_1, F_2)*, two fault models $m_{\text{fail},1}$ and m_{fail2} are created. The first differs from m_{ok} in replacing each requirement R_i with consequence

$$B = \left(\left[F_1\right]_{ta}^{tb} \wedge c_B\right)$$

by $R_{i,\text{fault}}$ with:

$$R_{i,\text{fault}} = R_i\left[B \rightarrow B'\right],$$
$$B' = \left(\left[\left(F_1 \wedge \neg F_2\right) \vee \left(\neg F_1 \wedge F_2\right)\right]_{ta}^{tb} \wedge c_B\right).$$

Replacing each F_1 in the formulas above by F_2 and vice versa leads to the second model $m_{\text{fail},2}$.

Note that all fault types above refer to the structure of the requirements stated as templates.

There are more plausible fault types, and we list some of them:

Unwanted Temporal Ordering Dependency

Contrary to the specification, if the events of the start condition occur in a specific order, the consequence of a requirement does not occur.

Example: The specification contains the requirement „When buttons A and B are both down for 5 seconds, then X must occur immediately" According to this requirement the consequence must occur independently of the order of pressing A and B.

Wrong Requirement Priority

Contrary to the specification, a requirement has a lower priority than another requirement (if a requirement has a higher priority than another, it over-writes it, i.e. in a condition matching both start conditions, only the consequence of the requirement with the higher priority if they are contradictory).

Example: assuming the following two requirements, where the first is higher prioritized than the second. Requirement 1: "If the main switch is turned to position ‚off', the system must immediately turn off itself". Requirement 2: "if the pedal is released in mode $active_1$, the system must immediately switch to mode $active_2$". Here a Wrong Requirement Priority Fault exists, if the system switches to $active_2$ instead of deactivating itself, if the pedal and the switch are pressed simultaneously in $active_1$.

Requirement Violated in Temporary States

In a specific temporary state the consequence of a requirement does not hold.

Example: assume in some situations mode m_1 is active for 500ms only and then becomes inactive without further interaction. The fault type states that the consequence of some selected or all applicable requirements do not hold during m_1 although the start-conditions are fulfilled. The underlying hypothesis is that such short states may be easily overlooked during testing.

Incorrect Bracketing

Wrong brackets in a state expression

For instance, the text "…button A or button B and button C is pressed…" in a requirement document could be interpreted as (A∨B)∧C is pressed or A∨(B∧C) is pressed. This fault type assumes that the wrong interpretation was chosen for implementation.

Note that for some fault types additional knowledge not contained in the requirement specification is needed, e.g. for an Unwanted Resource Shortage Fault, information about dependencies between facts (or requirements) and the resource are needed.

Note that a faulty requirement may contradict other requirements, thus it has to override them in order to get a faulty but consistent specification (e.g. by assigning each requirement a priority).

In summary, we tried to illustrate in this section that

- there exist intuitive concepts of fault types that motivate tests,
- these fault types are obtained as transformations of the respective requirements (and that this re-

quires the preservation of the structure of the requirements)
- they can be stated in the same language as the requirements, FRL.

6. Input and Output of Test Generation

Based on the described foundations, a test generation specification TGS contains all information, besides the specification $model_{ok}$ of the system-under-test, needed to successfully perform fault-based test generation. A TGS is represented as a tuple:

$TGS = (TESTBENCH, SET_{TESTIDEAS})$,

where

$TESTBENCH = (FACTS_{obs}, FACTS_{causal})$

describes the attributes of the test bench, which is the technical interface to the system-under-test. When switching to another test bench, TESTBENCH must be adopted properly. It declares the set $FACTS_{obs}$ of facts observable and the set $FACTS_{causal}$ of facts that can be manipulated by the tester. Optionally the test bench's maximal temporal resolution of the observations, Δt_{obs}, and of the stimuli, Δt_{obs}, may be stated. For instance, Δt_{obs} =10ms stating that events lasting less than ten milliseconds are not visible, becomes handy if the stimuli is observed only by a human tester.

The set $SET_{TESTIDEAS}$ consists of test ideas

$TESTIDEA = (TYPE_{fault}, REQUIREMENT)$.

A test idea specifies that the fault type $TYPE_{fault}$ has to be applied to REQUIREMENT, which results in one or several fault models. Fault specific parameters, such as a reference to resource sharing for Resource Shortage Faults, may be given in addition.

Since the test generation algorithm produces tests in order to discriminate the OK behavior from the various fault models, the purpose of the test can be explained to the test engineer by referring to a specific requirement and to certain fault types, i.e. at a conceptual level he is familiar with. One could also try to display the hypothesized fault types in terms of templates. Because there is no guarantee that the FRL fault model corresponds to any of the templates offered for requirement acquisition, this would only work if such templates are generated automatically from FRL.

7. Future Work

The project described here is driven by the application requirements. Therefore, we have to avoid overloading the acquisition of the requirements and the domain theory. This is why we have to allow a rather coarse level for expressing FACTS, with the obvious drawback that at this propositional level many of the interrelations of various requirements, including inconsistency, refinement, redundancy, remain implicit and cannot be exploited by the algorithm, which weakens its results. Such interrelations can be made explicit in the domain theory, causing higher efforts on this side. The alternative is to allow for a more structured representation. An obvious extension is to introduce variables and assignment of values or ranges to them. Currently, we are performing an initial evaluation, which will provide us with feedback from the test engineers and with hints on an appropriate trade-off between limiting efforts on the requirement acquisition side and the quality and utility of the generated tests. At least, we will be able to demonstrate what can be gained by a more structured and systematic requirement acquisition.

Acknowledgements

Thanks to the Model-based Systems and Qualitative Modeling Group at the Technical University of Munich, Oskar Dressler, Martin Sachenbacher, and the reviewers for their helpful comments. We also thank Audi AG, Ingolstadt, and, in particular, Reinhard Schieber for support of this work.

References

[Boot et al. 99] Boot, R., Richert, J., Schutte H. and Rukgauer, A. 1999, *Automated test of ECUs in a hardware-in-the-loop simulation environment*. Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design.

[Chaochen-Hansen 03] Chaochen, Z., and Hansen, M. R. 2003, *Duration Calculus. A Formal Approach to Real-Time Systems*. Springer.

[Esser-Struss 07] Esser, M. and Struss, P. 2007, *Fault-model-based Test Generation for Embedded Software*, IJCAI2007, Hyderabad, India.

[Fuchs-Schwitter 96] Fuchs, N. E., Schwitter, R. 1996, *Attemp to Controlled English (ACE)*, CLAW 96, First International Workshop on Controlled Language Applications, University of Leuven, Belgium

[Struss 94] Struss, P. 1994, *Testing Physical Systems.* In Proceedings of AAAI-94, Seattle, USA.

Generating Manifestations of Max-Fault Min-Cardinality Diagnoses

Alexander Feldman[1] and **Gregory Provan**[2] and **Arjan van Gemund**[1]
[1]Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {a.b.feldman,a.j.c.vangemund}@tudelft.nl
[2]University College Cork, Department of Computer Science, College Road, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: g.provan@cs.ucc.ie

Abstract

Computing test vectors that are optimized to isolate faults is an important area of diagnostics. The literature has focused on test vectors for single-fault diagnoses. This article generalizes this, addressing the problem of computing Max-Fault Min-Cardinality (MFMC) observation vectors in Model-Based Diagnosis (MBD), and proposing two algorithms for solving it. An MFMC observation vector is an observation that, for a given system description, results in the maximum number of faults in the minimum cardinality diagnosis. Computing MFMC observation vectors has application in testability analysis, MBD benchmarking, optimal sensor placement and other areas of model-based reasoning. We discuss the high computational complexity of the MFMC problem and introduce stochastic methods to reduce the solution complexity. The first method for computing MFMC is based on importance sampling while the second one is based on simulated annealing. Both algorithms lead to significant speed-up compared to exhaustive search and perform best for different classes of system descriptions. The algorithms described in this paper have been implemented and tested on a benchmark of combinatorial circuits.

Introduction

The problem of computing minimum cardinality diagnoses, given an observation and a system description, is central to Model-Based Diagnosis. Consider the inverse problem of computing an observation which distinguishes a specific number of faulty components. Computing observations that distinguish a single failing component is studied by Automatic Test Pattern Generation (ATPG) and dates back to (Roth 1966). The goal of ATPG is, then, to compute a *sequence* of test vectors that can distinguish every possible single fault in a device.

Single-fault ATPG has been extended to finding observation vectors leading to double faults (Hughes 1988) and to multiple faults (Kubiak & Fuchs 1991). All of these approaches have several drawbacks, including: (1) they are inherently suboptimal, i.e., they do not answer the question of what is the maximum number of faults distinguishable by a single test vector, (2) they suffer from very high computational complexity, and (3) they severely limit the class of system abstractions by using various simulation techniques.

Few papers address algorithms computing observation vectors that distinguish the *maximum* number of failing components in a system, e.g., (Abramovici 1981). One can devise a class of more general algorithms for this, based on techniques from MBD and abductive reasoning. This paper formalizes the problem of finding Max-Fault Min-Cardinality (MFMC) observation vectors and proposes two algorithms for solving it. These two methods are very efficient, given the existence of a fast MBD engine.

One of the advantages of the algorithms in this paper over the related *k-fault* ATPG algorithms is that they do not impose *any limitations* on the model (e.g., they do not require stuck-at modes or unlimited observability). This makes them applicable not only to testing but to a wider range of Model-Based Reasoning problems. The MFMC algorithms can be used for MBD benchmarking (Provan & Wang 2007), optimal sensor placement (Console, Picardi, & Ribaudo 2000) and other applications.

A summary of our contributions follows. This paper introduces the MFMC problem and two algorithms for computing MFMC observation vectors. The first one is based on importance sampling and the second one on simulated annealing. The two algorithms are empirically analyzed on a number of diagnostic models. Furthermore, we discover some properties of the MFMC search and reason about its computational complexity.

The rest of this paper is organized as follows. The section which comes after this introduction defines the basic MFMC framework. It is followed by a short discussion on some complexity issues. The fourth section suggests algorithms for solving the MFMC problem. Finally, we evaluate the empirical performance of the algorithms.

Preliminaries

The discussion starts by formalizing some basic notions in MBD, extending the notions in (de Kleer, Mackworth, & Reiter 1992). A *model* of an artifact is represented as a propositional **Wff** over some set of variables V. Discerning a subset of them as *assumable* or *observable* gives us a diagnostic system.

Definition 1 (Diagnostic System)**.** A diagnostic system DS is defined as the triple $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, where SD is a propositional theory describing the behavior of the system, COMPS is a set of assumable variables in SD, and OBS is a set of some observable variables in SD.

Although it is not strictly necessary, throughout this paper we will assume that $\text{OBS} \cap \text{COMPS} = \emptyset$.

A Running Example

The simple Boolean circuit shown in Figure 1 is used to illustrate the notions in this papers and the workings of the algorithms we propose. The 2-to-4 line demultiplexer consists of four Boolean inverters and four and-gates.

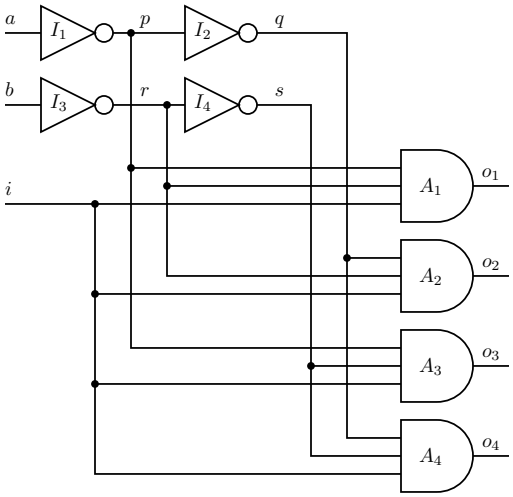


Figure 1: A 2-to-4 line demultiplexer.

The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models an inverter, where the variables i, o, and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2 \wedge i_3)$. These propositional formulae are copied for each gate in Figure 1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus generating the following propositional formula for SD:

$$\text{SD} = \begin{cases} I_1 \Rightarrow (a \Leftrightarrow \neg p) \\ I_2 \Rightarrow (p \Leftrightarrow \neg q) \\ I_3 \Rightarrow (b \Leftrightarrow \neg r) \\ I_4 \Rightarrow (r \Leftrightarrow \neg s) \\ A_1 \Rightarrow (o_1 \Leftrightarrow i \wedge p \wedge r) \\ A_2 \Rightarrow (o_2 \Leftrightarrow i \wedge r \wedge q) \\ A_3 \Rightarrow (o_3 \Leftrightarrow i \wedge p \wedge s) \\ A_4 \Rightarrow (o_4 \Leftrightarrow i \wedge s \wedge q) \end{cases}$$

The set of component (assumable) variables is $\text{COMPS} = \{I_1, \ldots, I_4, A_1, \ldots, A_4\}$. The set of observable variables is $\text{OBS} = \{i, a, b, o_1, \ldots, o_4\}$.

Diagnostic Framework

The traditional query in MBD results in finding terms of assumable variables which are explanations for the system description and an observation. The first definition of diagnosis uses a set notation.

Definition 2 (Diagnosis)**.** A *diagnosis* for the system $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$, given an observation term α over variables in OBS, is a set $D \subseteq \text{COMPS}$ such that:

$$\text{SD} \wedge \alpha \wedge \left[\bigwedge_{c \in D} \neg h_c \right] \wedge \left[\bigwedge_{c \in (\text{COMPS} \setminus D)} h_c \right] \not\models \bot$$

Under *lex parsimoniae* we are interested in computing those diagnoses that are not subsumed by other diagnoses of $\text{SD} \wedge \alpha$.

Definition 3 (Minimal Diagnosis)**.** A diagnosis D is minimal iff no proper subset $D' \subset D$ exists, such that D' is also a diagnosis.

Throughout this paper we interchangeably use a propositional notation for expressing diagnoses. In this case we simply construct a conjunction of literals, each literal having a negative sign if its respective variable is in D and a positive sign otherwise. Consider the example from Figure 1 and an observation $\alpha = a \wedge b \wedge i \wedge \neg o_4$. In this case $D_1 = \{I_1, I_2\}$ is a diagnosis and $D_2 = \{I_1\}$ is a minimal diagnosis (there are four more minimal diagnoses for $\text{SD} \wedge \alpha$). Alternatively, instead of D_1 we may write $D'_1 = \neg I_1 \wedge \neg I_2 \wedge I_3 \wedge I_4 \wedge A_1 \wedge \ldots \wedge A_4$.

The *cardinality* of a diagnosis D is the size of D and is denoted as $|D|$. It represents the number of faulty components in COMPS given SD and α. Next to computing minimal diagnoses, it is of interest to MBD to compute some or all minimal-cardinality diagnoses, given a diagnostic problem.

Definition 4 (Minimal-Cardinality Diagnosis)**.** A diagnosis D is a minimal-cardinality diagnosis if it is a minimal diagnosis and no other diagnosis D' exists such that $|D| < |D'|$.

The cardinality of a minimal-cardinality diagnosis computed from a system description SD and an observation α is denoted as $\mathit{MinCard}(\text{SD} \wedge \alpha)$. For our example and the observation $\alpha = a \wedge b \wedge i \wedge \neg o_4$, it follows that $\mathit{MinCard}(\text{SD} \wedge \alpha) = 1$. Note that in this case all minimal diagnoses are also minimal-cardinality diagnoses.

There are minimal diagnoses which are not minimal-cardinality diagnoses. Let us consider, for example, the diagnostic system $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$, where $\text{SD} = (h_1 \wedge h_2 \wedge x) \vee (h_4 \wedge x)$, $\text{COMPS} = \{h_1, h_2, h_3, h_4\}$, $\text{OBS} = \{x\}$, and an observation $\alpha = x$. In this case, $D_1 = \{h_1, h_2, h_3\}$ is a non-minimal diagnosis, $D_2 = \{h_1, h_2\}$ and $D_3 = \{h_4\}$ are minimal diagnoses, but only D_3 is a minimal-cardinality diagnosis.

Definition 5 (MFMC Observation)**.** Given a system description SD, a Max-Fault Min-Cardinality (MFMC) observation is an instantiation α over variables in OBS such that $\mathit{MinCard}(\text{SD} \wedge \alpha)$ is maximized.

Throughout this paper we will use the term *MFMC diagnosis*, that is any of the minimal-cardinality diagnoses D for which α is an MFMC observation. The cardinality of the MFMC diagnosis of a diagnostic system DS is denoted as $\mathit{MaxCard}(\text{DS})$.

The MBD literature often considers a class of theories that define normative behavior of their components only, i.e., models which specify no fault-modes. These models are sometimes referred to as *weak-fault models*, or *ignorance of abnormal behavior* (de Kleer, Mackworth, & Reiter 1992), or, in our case, *implicit fault systems*.

Definition 6 (Implicit Fault System)**.** A diagnostic system DS belongs to the class **IFS** iff SD is in the form $(h_1 \Rightarrow F_1) \wedge \ldots \wedge (h_n \Rightarrow F_n)$ such that for $1 \leq i, j \leq n$, $\{h_i\} \subseteq$ COMPS, $F_j \in$ **Wff**, and none of h_i appears in F_j.

Traditionally diagnosis and minimal diagnosis are defined only in the context of implicit fault systems. Note that a diagnosis assigns values to all variables in COMPS.

A stronger notion of diagnosis exists for systems that impose no restriction on the propositional theory (e.g., strong-fault models). To introduce these types of diagnoses we will borrow the next definition from (Darwiche 1998).

Definition 7 (Consequence)**.** Given a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{OBS}, \mathrm{COMPS} \rangle$, and an observation α, the consequence of $\mathrm{SD} \wedge \alpha$, is a sentence $Cons(\mathrm{SD} \wedge \alpha)$, such that all its literals are in COMPS, $\mathrm{SD} \wedge \alpha \models Cons(\mathrm{SD} \wedge \alpha)$ and for any term β, $\mathrm{SD} \wedge \alpha \models \beta$ if $Cons(\mathrm{SD} \wedge \alpha) \models \beta$.

The next definition gives us another way to represent diagnosis and a more expressive explanation of $\mathrm{SD} \wedge \alpha$.

Definition 8 (Partial Diagnosis)**.** Given a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{OBS}, \mathrm{COMPS} \rangle$, and an observation α, a partial diagnosis ω is defined as a term over the set of assumable variables $h \in$ COMPS, such that $\omega \models Cons(\mathrm{SD} \wedge \alpha)$.

By distinguishing these partial diagnoses only, which are not contained in other implicants of $Cons(\mathrm{SD} \wedge \alpha)$, we get partial diagnoses which are minimal under subsumption. Computing these diagnoses, however, is strictly more difficult than computing the set of all minimal diagnoses.

Definition 9 (Kernel Diagnosis)**.** A partial diagnosis ω is a kernel diagnosis iff no partial diagnosis ω' exists, such that ω' is the conjunction of a proper subset of the literals in ω.

Consider again the example from Figure 1 and an observation $\alpha = a \wedge b \wedge i \wedge \neg o_4$. In this case $\omega_1 = \neg I_1 \wedge \neg I_2$ is a partial diagnosis and $\omega_2 = \neg I_1$ is a kernel diagnosis. These are similar to the results for diagnosis and minimal diagnosis, but consider changing the models of all inverters in SD from $h \Rightarrow (o \Leftrightarrow \neg i)$ to $[h \Rightarrow (o \Leftrightarrow \neg i)] \wedge (\neg h \Rightarrow \neg o)$. In the latter scenario, diagnoses and minimal diagnoses are not defined and $\omega_1 = I_1 \wedge \neg I_2 \wedge I_3$ is a kernel diagnosis. Note that, for example, $\omega_2 = \neg I_1 \wedge \ldots \wedge \neg I_4$ is not a partial diagnosis even though it contains a superset of the faulty components in ω_1.

The cardinality of a partial diagnosis ω, denoted as $Card(\omega)$, is defined as the number of negative literals in ω. In the above example, $Card(\omega_1) = 1$. In a similar way we derive minimal-cardinality partial diagnosis and MFMC partial diagnoses. It can be shown, that if $\mathrm{SD} \in$ **IFS** the MFMC cardinalities would be the same for both kernel and minimal diagnoses.

Our methods for computing MFMC observation vectors rely on a diagnostic oracle. This oracle is supplied with a system description SD and a candidate observation α. Depending on the implementation of this oracle, our algorithms will compute MFMC observation vectors for either minimal diagnoses or partial diagnoses. For the rest of this paper, when it is clear from the context or from the model, we will drop the qualification of the diagnosis type.

Before we proceed with the algorithmic sections, we can truncate the search space for MFMC observation vectors by realizing that, for finding $MaxCard(\mathrm{DS})$, it is necessary to instantiate all observable variables in SD.

Proposition 1 (Observation Monotonicity)**.** *Given that* SD *is a system description and α and β are two observations such that $\alpha \supseteq \beta$ then it holds that $MinCard(\mathrm{SD} \wedge \alpha) \geq MinCard(\mathrm{SD} \wedge \beta)$.*

Proof. The proof comes directly from the definitions of diagnosis. We construct a system of Boolean equations B in the following manner. First, the propositional **Wff** in SD is converted to a Boolean equation in a straightforward manner and the latter is added to B. Second, for each literal $l_i \in \alpha$, an equation of the form $l_i = 1$ or $l_i = 0$ (depending on the polarity of l_i) is appended to B. A system of Boolean equations B' is constructed from SD and β in an analogous way. The solutions of B and B' are the implicants of $\mathrm{SD} \wedge \alpha$ and $\mathrm{SD} \wedge \beta$, respectively. Observe, that, due to the fact that $\alpha \supseteq \beta$, the equations in B' are a superset of these in B and both are over the same set of variables. But $S(B') \leq S(B)$, where $S(X)$ denotes the number of solutions in a system X. The above holds also when the solutions of B and B' are ordered according to their cardinality. Hence, if a diagnosis with a cardinality smaller than the smallest cardinality diagnosis in B' exists, it is in B. □

Next we discuss some computational complexity properties of generating MFMC observation vectors.

Complexity of the MFMC Problem

We have already seen that there is dependency between the observability of a model and the cardinality of the MFMC diagnosis (the average complexity of the problem). Before we continue our reasoning with the worst-case complexity of MFMC, we motivate the need of MFMC by noting that building diagnosable systems is expensive in terms of sensors (observables).

Assume that we have a system with k binary-valued sensors and n components, and that each component can be either faulty or healthy, i.e., the fault description does not define failure modes. We define a failure as an instantiation of fault modes, and a test as an instantiation of sensors.

There are 2^k test-vector settings (the test space Γ), and $2^n - 1$ possible fault combinations (the failure space Ω).

The ability to isolate all fault combinations (in which case the system is diagnosable) is typically impossible for practical reasons. In most cases, for a large system, it is too expensive to provide enough sensors to ensure full diagnosability. The following theorem defines the number of sensors needed to isolate q failures:

Theorem 1. *The minimum number of sensors needed to isolate q failures is given by $\lceil log_2(q+1) \rceil$.*

Proof. If we have q binary sensors, then there are 2^q distinct sensor signatures, of which $\{0, ..., 0\}$ is the nominal signature. Hence $2^q - 1$ signatures denote distinguished failures. This can be rearranged simply as follows:

$$
\begin{aligned}
q \text{ sensors} &\rightsquigarrow 2^q - 1 \text{ distinguished failures} \\
q+1 \text{ sensors} &\rightsquigarrow 2^q \text{ distinguished failures} \\
\lceil log_2(q+1) \rceil \text{ sensors} &\rightsquigarrow q \text{ distinguished failures}
\end{aligned}
$$

□

As a consequence, we must operate in a world where some failures are indistinguishable given Γ (they mask each other). In this article we choose to focus on the probabilistically most-likely failures, assuming that we have a probability distribution over the individual faults and all faults are mutually independent.

To complete our theoretical notions, we reason about the worst-case complexity of the MFMC problem. We can show that solving a simplified restriction of the MFMC problem is NP-complete.

Theorem 2 (Complexity of Restricted MFMC)**.** *Given a diagnostic model* $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{OBS}, \mathrm{COMPS} \rangle$ *and a diagnosis D, it is NP-complete to determine a manifestation for the observation* α.

This theorem can be proven by observing that it is just the dual to the problem of determining the existence of a diagnosis D given the observation α (Theorem 4.7 of (Bylander *et al.* 1991).)

Further, the complexity of the MFMC problem is likely to be higher than that of isolating multiple-fault diagnoses (which is NP-complete (Bylander *et al.* 1991; Friedrich, Gottlob, & Nejdl 1990)), or that of computing a minimum-size test set to isolate all single stuck-at faults in electronic circuits (which is NP-complete (Krishnamurthy & Akers 1984)). With regard to Multiple-Fault Diagnosis (MFD), MFMC introduces an optimization task that makes multiple calls to an MFD oracle, which is clearly harder. With regard to testability analysis, MFMC addresses the multiple-fault case, and is applicable to arbitrary models, and not just stuck-at circuit models.

It is also likely that approximating MFMC within a constant factor is intractable. Approximating the *single-fault* minimum-size test set within a factor $\delta > 1$ of optimality is NP-hard (Krishnamurthy & Akers 1984).

As a consequence of intractability and other practical issues, such as dealing with failures which are indistinguishable (they mask each other), we focus on the probabilistically most-likely failures, assuming that we have a probability distribution over the individual faults and all faults are mutually independent.

Algorithms for Computing MFMC

In this section we discuss algorithms for computing MFMC. The first one is based on exhaustive search, hence it is suitable for understanding the basics of the MFMC computation only. The second algorithm borrows from Importance Sampling (IS) (Yuan & Druzdzel 2006) to skip over health assignment leading to faults of low cardinality. Finally, we suggest a simulated annealing algorithm for generation of MFMC.

A Naïve Brute-Force Algorithm

Obviously the model of the 2-to-4 line demultiplexer belongs to **IFS**. An algorithm which finds such an observation by trying all possible instantiation of the observable variables is shown in Algorithm 1.

Algorithm 1 An exhaustive search algorithm for generation of MFMC observation vectors.

```
1: function NaïveMFMC(SD, OBS) returns a term
       inputs: SD, a propositional theory
               OBS, a set of observable variables
       local variables: α, ω, R, terms
                        M, an integer, initially 0
2:     for all α ← INSTANTIATE(OBS) do
3:         ω ← FINDMCDIAG(SD ∧ α)
4:         if M < COUNTFAULTS(ω) then
5:             R ← α
6:             M ← COUNTFAULTS(ω)
7:         end if
8:     end for
9:     return R
10: end function
```

The outer loop of Algorithm 1 tries all the $2^{|\mathrm{OBS}|}$ instantiations of the variables in OBS. For each possible instantiation α it finds the minimal cardinality diagnosis by issuing a call to the FINDMCDIAG subroutine. The observation leading to a diagnosis with a maximum number of faults (in this example COUNTFAULTS simply counts the number of negative assumable literals in ω) is preserved as a result.

Any method for computing a diagnosis can be used as an implementation of FINDMCDIAG and various methods like compilation (Darwiche 2001) or heuristics and conflict exploitation (Williams & Ragno 2004) can be used to speed-up this function.

For the circuit shown in Figure 1, Algorithm 1 exhaustively generates 128 instantiations of OBS. For example, consider the arbitrary assignment $\alpha = \neg a \wedge \neg b \wedge i \wedge \neg o_1 \wedge \ldots \wedge \neg o_4$. Clearly, the only minimal-cardinality diagnosis of $\mathrm{SD} \wedge \alpha$ is $\omega = \neg A_1$, and $Card(\omega) = 1$.

For the example demultiplexer, $MaxCard(\mathrm{SD}) = 4$ and there is a total of 4 observation vectors that can discern minimal-cardinality diagnosis of 4 faults. One of these max-fault min-cardinality observation vectors is $\alpha_{mfmc} = \neg a \wedge \neg b \wedge \neg i \wedge o_1 \wedge \ldots \wedge o_4$. Interestingly, from all the 128 possibilities, there are 8, 40, 53, and 23 observation vectors leading to a nominal, single-fault, double-fault and triple-fault diagnosis respectively.

Computing MFMC Approximations through Importance Sampling

Our IS algorithm is based on a weighted approach and works for a restricted set of system descriptions. Furthermore, the algorithm uses an extension to DS, a valuation function $Pr : \mathrm{COMPS} \rightarrow [0, 1]$. In practice, Pr assigns *a priori* probabilities to the system failure modes.

We assign an *a priori* valuation to an assignment α using $Pr(\alpha) = \Pi_{x \in \alpha \cup \mathrm{COMPS}} Pr(x)$, which corresponds to

assuming that all variables in SD are conditionally independent.

Consider systems that model the faulty behavior of their components. For some of them, it is possible to partition the set of observable variables OBS into two subsets IN and OUT (denoting input and output variables respectively), and after giving values to IN and COMPS, to use a reasoning algorithm (e.g., unit-propagation) to find a *unique assignment* to the values in OUT.

Definition 10 (Explicit Fault System)**.** Given a system DS and a partitioning $\text{OBS} = \text{IN} \cup \text{OUT}$, $\text{DS} \in \textbf{EFS}$ if for any instantiation ϕ of all variables in $\text{IN} \cup \text{COMPS}$, it holds that there is exactly one term ψ such that $\phi \models \text{SD} \wedge \psi$ and ψ is an instantiation of all variables in OUT.

The above restriction on the class of the propositional models allows us to introduce the algorithm which follows.

Algorithm 2 An algorithm for computing an MFMC approximations by using Importance Sampling.

```
1: function ISOBS(DS, IN, Pr, Pr*) returns a term
       inputs: DS = ⟨SD, COMPS, OBS⟩, a diag. system
               IN, a set of variables, IN ⊆ OBS
               Pr, a valuation function
               Pr*, a biasing pdf
       parameters: S#, integer, number of samples
       local variables: i, o, h, ω, R, terms
                        M, s, integers, initially 0
2:     while s < S# do
3:         ⟨h, i⟩ ← INSTANTIATE(COMPS, IN, Pr, Pr*)
4:         o ← PROPAGATE(SD ∧ i ∧ h)
5:         ω ← FINDMCDIAG(SD ∧ i ∧ o)
6:         if M < COUNTFAULTS(ω) then
7:             R ← i ∧ o
8:             M ← COUNTFAULTS(ω)
9:         end if
10:        s ← s + 1
11:    end while
12:    return R
13: end function
```

Algorithm 2 uses *simulation* to compute a subset of all (physically) possible states of a system. A biasing probability density function (pdf) is used to increase the probability of a sample being consistent with a higher number of faults. For each of the $S_{\#}$ samples over the input values i, the outputs o are computed and the minimal diagnosis consistent with $i \wedge o$ is computed. The observation which leads to a maximum number of faults is preserved throughout the sampling process and returned at the end of the procedure.

We discuss the implementation of the INSTANTIATE function which determines the quality of the observation vectors and the performance of the algorithm. Assuming equal probabilities for the input variables, it implements the function given next for assigning values to the variable set supplied as an argument.

$$P(x = \textbf{True}) = \begin{cases} k\,[1 - Pr(x)] & : \quad x \in \text{COMPS} \\ 0.5 & : \quad x \in \text{OBS} \end{cases}$$

The above function uses a skewing coefficient k to scale the probability of an assumable variable being instantiated as faulty. In general, this coefficient depends on the model and we will observe its effect in the experimentation section.

The implementation of the PROPAGATE subroutine is straightforward. We suggest the use of a Binary Constraint Propagation (BCP) method which can efficiently derive a satisfying assignment for the output variables. The auxiliary function GETOBS is used to discern these literals in a model of SD which instantiate observable variables.

Algorithm 2 computes minimal cardinality diagnosis by issuing a call to the FINDMCDIAG subroutine which is the same as in Algorithm 1. The observation leading to a diagnosis with a maximum number of faults (in this example COUNTFAULTS simply counts the number of negative assumable literals in ω) is preserved as a result. The number of calls to the potentially most expensive subroutine FINDMCDIAG decreases from $2^{|\text{OBS}|}$ (in the case of an exhaustive search) to $S_{\#}$, where $S_{\#}$ is generally low.

To illustrate the workings of Algorithm 2 on the 2-to-4 line demultiplexer we have introduced in the running example, it is necessary to change the system description used by the exhaustive search algorithm. The reason for this is that due to the weak-fault model the propagation routine would not be able to derive the values of the output variables given all inputs and health. In order to fix this, instead of one we use two assumable variables per logic-gate f_0 and f_1 to denote "stuck-at-zero" and "stuck-at-one" respectively.

The new formula for modeling an inverter is $(f_0 \Rightarrow \neg o) \wedge (f_1 \Rightarrow o) \wedge (\neg f_0 \wedge \neg f_1 \Rightarrow (\neg i \Leftrightarrow o)) \wedge (\neg f_o \vee \neg f_1)$. Each of the and-gates is represented as $(f_0 \Rightarrow \neg o) \wedge (f_1 \Rightarrow o) \wedge (\neg f_0 \wedge \neg f_1 \Rightarrow (i_1 \wedge i_2 \wedge i_3 \Leftrightarrow o)) \wedge (\neg f_o \vee \neg f_1)$. Again, we have to rename the variables for each gates, receiving a system containing eight Boolean equations. For brevity, we will omit the actual system description from this paper.

In running Algorithm 2 on the demultiplexer circuit, we assign the set of inputs $\text{IN} = \{a, b, i\}$, the set of output variables $\text{OUT} = \{o_1, o_2, o_3, o_4\}$, the Pdf $Pr(f_0 = \textbf{True}) = Pr(f_1 = \textbf{True}) = 0.01$ and $k = 25$. The number of samples $S_{\#}$ has been set to 25.

For the demultiplexer circuit, Algorithm 2 works as follows. First it draws random values with equal probabilities for the input variables a, b and i. Then it draws values for the "stuck-at-zero" and "stuck-at-one" assumables with probability 0.25. After propagation, the values for the output variables $o_1, \ldots, o_4$ are computed. At the end, from all samples the observation consistent with minimal-cardinality diagnosis of maximum number of faults is chosen. For this example run, let this observation be $\alpha_{is} = a \wedge \neg b \wedge \neg i \wedge o_1 \wedge \neg o_2 \wedge o_3 \wedge o_4$. As the reader can see, this is consistent with a minimal cardinality triple-fault diagnosis $\omega_{\text{is}} = \big(A_1 \equiv F^1\big) \wedge \big(A_3 \equiv F^1\big) \wedge \big(A_4 \equiv F^1\big)$, where the proposition $A_n \equiv F^1$ denotes an and-gate "stuck-at-one". The received observation leads to a diagnosis of acceptable quality, but the sampling continues until $S_{\#} = 25$. A higher number of samples increases the probability of finding an observation leading to a quadruple fault which is the optimum for this example.

A Simulated Annealing Algorithm

Algorithm 3 has no restrictions on the theories for which it can compute MFMC approximations. The technique it employs is simulated annealing (Rutenbar 1989).

Algorithm 3 A simulated annealing algorithm for generation of MC observation vectors of multiple faults.

```
1: function MCHILLCLIMB(SD, OBS) returns a term
       inputs: SD, propositional theory
               OBS, set of observable variables
       parameters: T_min, real, "cool-off" temperature
                   T_max, real, starting temperature
                   N, integer, number of tries
                   r, real, decay rate
       local variables: v_c, v_n, terms
                        t, j, ΔE, integers
                        T, real, current temperature
2:     t ← 0
3:     repeat
4:         v_c ← INSTANTIATERANDOM(OBS)
5:         j ← 0
6:         repeat
7:             T = T_max e^{-jr}
8:             for all v_n ← FLIPOBSERVABLE(v_c) do
9:                 ΔE ← f(SD ∧ v_n) − f(SD ∧ v_c)
10:                if ΔE > 0 then          ▷ Better MFMC?
11:                    v_c ← v_n           ▷ Accept the move.
12:                else                    ▷ Consider going downhill.
13:                    if RAND() < e^{T^{-1} ΔE} then
14:                        v_c ← v_n
15:                    end if
16:                end if
17:            end for
18:            j ← j + 1
19:        until T < T_min
20:        t ← t + 1                       ▷ Number of attempts.
21:    until t = N
22:    return v_c
23: end function
```

Algorithm 3 performs a maximum number of N independent attempts, each one starting from a random observation vector. These random observations are returned by INSTANTIATERANDOM, which assigns with equal probability **True** or **False** to each observable. As we will see in the experimentation section, a random observation vector is most likely to lead to a diagnosis of cardinality $M/2$, where M is the number of faults in the MFMC diagnosis.

The algorithm manipulates the initial random observation in an attempt of reaching a good optimum. The manipulation of the observation vector, aiming at "hill climbing" is performed by the FLIPOBSERVABLE subroutine. The idea is to try "flipping" variables in the observation vector until a "flip" leads to an improvement in the fault cardinality. In some of the cases, however, "flipping" the value of an observable will lead to a decrease in the associated number of faults. In these cases Algorithm 3 considers accepting the "worse" observation in its current state v_c with some probability depending on the current temperature T. This is to allow the search to "escape", if stuck in a local optimum.

The probability of accepting a state v_n which is worse than the current one in v_c, defines the process of "cooling", which gives the name of Algorithm 3. The temperature T, which starts from $T_{\max}$ and decreases gradually to $T_{\min}$, results in such "worse" states being accepted with higher probability in the beginning of each iteration and decreasing the likelihood of such "flips" towards the end of the search, i.e., when the search "freezes".

The "value flips" are repeated until the current observation in v_c becomes consistent with a minimal-diagnosis fault of improved cardinality, computed by the evaluation function f. The implementation of f returns the number of faults in the minimal cardinality diagnosis consistent with SD $\wedge$ α and is the same as in Algorithm 3. In particular it calls COUNTFAULTS and FINDMCCARD.

The parameters of the simulated annealing algorithm which affect its performance and the quality of the MFMC observation vectors are $T_{\min}$, $T_{\max}$, N, and r. These are the starting and "cool-off" temperatures, the number of "tries" and the decay rate, respectively. Similar to (Spears 1996), we will choose $r = (|\text{OBS}| * N)^{-1}$. The rationale behind this choice of r is that we would like a "faster" decaying when the problem size or the number of random restarts increase. Increasing the temperature range $T_{\max} - T_{\min}$ or reducing the decay rate r would allow more thorough search to be performed from each randomly chosen position.

The RAND function returns a normally distributed random number x such that $0 \leq x < 1$. In the beginning of the decaying process T is close to 1, hence the search is stochastic, hence more likely to escape local optima. In the cooling process the search becomes like an ordinary hill-climbing.

Experimental Results

This section presents an empirical analysis of our algorithms.

Implementation Notes and Test Set Description

Our implementation is approximately 1000 lines of C code (excluding the diagnosis computation) and is a part of the LYDIA[1] package. Two variants of the critical subroutine for finding a minimal-cardinality diagnosis have been used. These utilize conflict-based search (Williams & Ragno 2004) and exploitation of structure (Feldman & van Gemund 2006). Despite the above state-of-the-art implementations, the diagnosis search routine constrains the efficiency of the MFMC observation vector search, which is not surprising knowing that we are trying to diagnose circuits with observations consistent with multiple-faults of large cardinality.

Table 1 summarizes the benchmark we have used for testing of our algorithms. All the models are derived from the 74XXX family of arithmetic circuits.

[1]This package for model-based fault diagnosis can be downloaded from `http://fdir.org/lydia/`.

Name	Description	V_w	H_w	O	C_w	C_s
74180	9-bit parity check	38	14	12	48	90
74139	2-to-4 decoders	42	18	14	52	106
74153	4-to-1 selector	44	16	14	62	110
74182	4-bit CLA	47	19	14	75	132
74283	4-bit adder	89	40	14	130	250
74L85	4-bit comparator	93	41	14	134	257
74181	4-bit ALU	138	62	22	216	402

Table 1: Basic characteristics of the fault models from the 74XXX circuits family.

Some of the basic properties of the models we have used for benchmarking are shown in Table 1. For the models belonging to **IFS**, V_w, H_w, and O are the total number of variables, the number of assumables $|\mathrm{COMPS}|$ and the number of observables $|\mathrm{OBS}|$, respectively. For the strong-fault models, used in the testing of Algorithm 2, the number of assumables is $H_s = 2H_w$, the number of observable variables is the same as in the weak-fault models and the total number of variables is $V_s = V_w + H_w$. Columns C_w and C_s show the number of clauses in the CNF representations of the weak and strong system descriptions, respectively.

All the experiments described in this paper are performed on a host with 1.86 GHz Pentium M CPU and 2 Gb of RAM.

MFMC Vector Sizes and Performance Results

A series of exhaustive MFMC experiments allowed us to make an interesting observation. For all the benchmark circuits we have tested, the empirical pdf of the MFMC fault cardinalities, in respect to the observation vectors, approximates a binomial pdf within a very small margin. The results for two of the circuits are plotted in Figure 2.

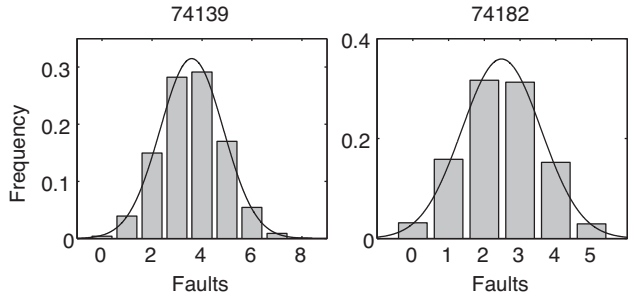


Figure 2: Empirical distributions of the fault cardinalities and normal pdf lines.

In the 74182 circuit, for example, given a randomly generated observation, the probabilities of a nominal kernel diagnosis or a quintuple fault are equal and very small. Analyzing the reasons for this behavior is a topic of its own, but our suggestion is that the underlying cause is the uncertainty introduced by the limited observability of the circuits (observability is defined as the ratio between the number of observable variables and all model variables).

The two implementations we test in this section are parameterized as follows. The importance sampling algorithm uses biasing coefficient $k = 25$ and the number of samples is equal to the number of components in the strong-fault model, i.e., $S_{\#} = H_s$. For the simulated annealing, we have set $T_{\min} = 0.1$, $T_{\max} = 0.105$, and $N = 4$.

Name	T_e	M	T_i	M_i	T_s	M_s
74180	1.4	2	0.04	2	0.1	2
74139	13.5	8	0.08	4.9	0.6	7.4
74153	8.8	2	0.08	2	0.2	2
74182	53.4	5	0.23	4.4	2.7	5
74283	371.9	5	5.74	3.9	18	3.9
74L85	196.9	3	3.28	3	14.5	3
74181	–	–	596.48	5.4	6114.4	6.5

Table 2: Fault cardinalities and wall-clock times [s] for computing of MFMC observation vectors.

The wall-clock times for the importance sampling and simulated annealing searches are shown in columns T_i and T_s of Table 2, respectively. The cardinalities of the MFMC observation vectors, computed by the two algorithms, are in M_i and M_s. As both MFMC computation methods in this paper are randomized, we have averaged the values of T_i, M_i, T_s, and M_s over 10 runs. It was possible to perform an exhaustive search, in all the test cases except for 74181. The results are shown in columns M and T_e, the former denoting the cardinality of the global MFMC optima and the latter – the computation time.

It is visible from Table 2 that Algorithm 2 outperforms Algorithm 3 by a factor of $2.5 - 11.7$. The cause of this is mainly in the smaller number of diagnoses which have to be computed, i.e., the importance sampling is more informed in reaching a local optimum. The observation vectors computed by Algorithm 2 lead to diagnoses of somewhat smaller cardinality than these computed by Algorithm 3. The difference is usually one fault, except in the 74139 circuit.

Figure 3 shows the progress of the MFMC search for two of the benchmark models. We note that for the 74139 circuit, the local optimum is found in the third iteration, while 74182 reaches its result from the first attempt. As a result decreasing N would keep the cardinality of the result but decrease the number of diagnostic computations.

From Figure 3 it is also visible that it is possible to climb to a good local optimum from an arbitrary initial random instantiation. This justifies the "observation bit flipping" operator (implemented in the FLIPOBSERVABLE subroutine of Algorithm 3) for climbing uphill in the stochastic search.

Conclusion

We have described two methods for computing MFMC observation vectors. The first algorithm, based on importance sampling, is applicable to a subset of all possible propositional theories, in particular to strong-fault models. This restriction, which does not exist in the simulated annealing

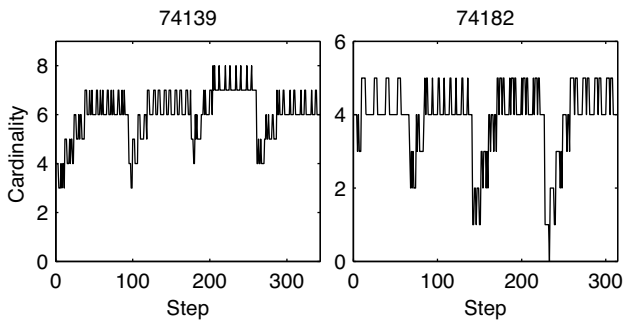


Figure 3: Fault cardinalities during sample simulated annealing sessions.

algorithm, results in a smaller number of calls to the underlying diagnostic engine and faster diagnostic reasoning.

We have studied the real-world behavior of the two algorithms on a series of combinatorial circuits. In all experiments, the number of faults in the diagnostic results was close to the global optimum: in some cases Algorithm 3 leads to diagnoses having one more fault than those computed by Algorithm 2.

An MBD oracle has been used in both of the algorithms. The only disadvantage of this is the underlying complexity of the diagnostic algorithms. Existing MBD heuristic methods, however, are tailored towards testing candidate diagnoses in order of likelihood. With the further development of the reasoning techniques, like the ones discussed in this paper, we expect new MBD heuristic methods to be developed that benefit from focusing the diagnostic search on high-cardinality diagnoses.

Finding MFMC observation vectors is of significant practical importance, and we expect more attention in the model-based reasoning community. Finally, we hope that the MFMC search will improve the algorithms for MBD, which in their turn will allow us to compute MFMC observations of better cardinality and for bigger models.

Acknowledgments

This work has been supported by STW grant DES.7015 and SFI grant 04/IN3/I524.

References

Abramovici, M. 1981. A maximal resolution guided-probe testing algorithm. In *Proc. DAC'81*, 189–195.

Bylander, T.; Allemang, D.; Tanner, M.; and Josephson, J. 1991. The computational complexity of abduction. *Artificial Intelligence* 49:25–60.

Console, L.; Picardi, C.; and Ribaudo, M. 2000. Diagnosis and diagnosability analysis using PEPA. In *Proc. ECAI'00*, 131–135.

Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3):197–222.

Feldman, A., and van Gemund, A. 2006. A two-step hierarchical algorithm for model-based diagnosis. In *Proc. AAAI'06*.

Friedrich, G.; Gottlob, G.; and Nejdl, W. 1990. Physical impossibility instead of fault models. In *Proc. AAAI*, 331–336.

Hughes, J. 1988. Multiple fault detection using single fault test sets. *IEEETCAD* 7(1):100–108.

Krishnamurthy, B., and Akers, S. B. 1984. On the complexity of estimating the size of a test set. *IEEE Trans. Computers* 33(8):750–753.

Kubiak, K., and Fuchs, W. K. 1991. Multiple-fault simulation and coverage of deterministic single-fault test sets. In *Proc. of the IEEE International Test Conference on Test*, 956–962.

Provan, G., and Wang, J. 2007. Automated benchmark model generators for model-based diagnostic inference. In *Proc. IJCAI'07*, 513–518.

Roth, J. P. 1966. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development* 10:278–291.

Rutenbar, R. A. 1989. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices* 5(1):19–26.

Spears, W. M. 1996. Simulated annealing for hard satisfiability problems. In *Second DIMACS Implementation Challenge: Cliques, Coloring and Satisfiability*, volume 26, 533–558.

Williams, B., and Ragno, R. 2004. Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*.

Yuan, C., and Druzdzel, M. J. 2006. Importance sampling algorithms for Bayesian networks: Principles and performance. *Mathematical and Computer Modelling* 43(9-10):1189–1207.

Interchange Formats and Automated Benchmark Model Generators for Model-Based Diagnostic Inference

Alexander Feldman[1] and **Gregory Provan**[2] and **Arjan van Gemund**[1]
[1]Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {a.b.feldman,a.j.c.vangemund}@tudelft.nl
[2]University College Cork, Department of Computer Science, College Road, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: g.provan@cs.ucc.ie

Abstract

This article proposes a Diagnosis Interchange Format (DIF), an XML-based interchange format for Model-Based Diagnosis (MBD). Its main purposes are to allow sharing of diagnostic models, observation data and fault hypotheses, and to facilitate empirical comparative study of the performance of existing and future MBD implementations. In this paper, we describe the syntax and the semantics of DIF as well as the principles underlying its design. Several examples are used to illustrate the use of DIF, with a particular focus on expressing structure, state and constraints for various domains. We also recommend several sources for creating a standardized MBD benchmark set and discuss possible extensions in subsequent versions of the format. We compare the proposed format to related approaches used in some modeling languages.

Introduction

The field of Model-Based Diagnosis (MBD) is in need of a repository of standardized models in order to test the efficiency of algorithms and the adequacy and efficiency of modeling representations. Algorithmic development in other AI disciplines (SAT, CSP, automated planning) has benefited from the existence of widely-accepted problem representations and benchmark sets. Since MBD covers a heterogeneous range of domains, ranging from discrete circuit through continuous-valued value dynamical systems like ecosystems, *standardizing* an MBD representation is a challenging task. This paper defines an interchange format for MBD, called the Diagnosis Interchange Format (DIF). We show how this model covers a wide range of model types, and compare DIF with languages for related purposes.

Our proposed language, DIF, can facilitate MBD research and technology transfer from both the perspective of algorithm development and of model representation.

From the algorithm development perspective, DIF can facilitate empirical comparative study of the performance of existing and future MBD implementations. To date, the lack of appropriate model- and algorithm-exchange formats has hindered such empirical comparisons. In fact, there has been no systematic study of the complexity of diagnosing real-world problems, and few good benchmarks exist to aid in such a study. Such a question is needed to answer the question of whether MBD is computationally difficult for the "average" real-world system; just worst-case results are known, such as the task of finding a kernel diagnosis of minimal cardinality being Π_2^P-complete (Eiter & Gottlob 1995). Given a standard model representation, the scarcity of existing benchmarks can be supplemented by automatically-generating models (Provan & Wang 2007) that have the properties of real-world models and conform to the MBD standard format.

From the representational perspective, DIF can provide model-interchange standards, thus overcoming difficulties in model sharing, which arise due to mutual incompatibility among the existing modeling representations, such as the Java-Based Model Programming Language (Williams & Nayak 1996), KOALA (Benazera, Travé-Massuyès, & Dague 2002), HYBRID CC (Carlson & Gupta 1998), LYDIA (Pietersma, Feldman, & van Gemund 2006).

We do not expect diagnostic reasoners to support the full Diagnosis Interchange Format (DIF) specification; e.g., a solver capable of reasoning in propositional logic would not support hybrid constraints. Rather, a diagnostic solver should specify the domain and constraint types it supports, the DIF specification would provide a model classification framework.

The rest of this paper is organized as follows. The next section discusses related work. The third section describes the syntax and semantics of DIF. Finally, we propose some initial benchmark problems and discuss future work.

Related Work

We now summarize a selection of formats that have influenced the design of DIF, and some model-generation tools of practical consideration.

Interchange Formats

An interchange format provides a standardized, declarative semantics, enabling the meaning of expressions in the representation to be understood without appeal to an interpreter for manipulating those expressions. Related formats include:

AI-ESTATE: The IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (Sheppard & Orlidge 1997) specifies observations, diagnoses, and model attributes common to most reasoners. It provides specialized model extensions in support of fault

tree and inference-based reasoners. The exchange format uses the EXPRESS info modeling language. Future versions are planned to support Bayesian inference models and XML data exchange.

AI-ESTATE is domain-independent, and is not limited to automatic testing and diagnosis but provides interfaces to manual testing. While being very comprehensive, the focus of AI-ESTATE is on interoperability as opposed to benchmarking MBD algorithms.

DiagML: The Diagnostic Modeling Language (Gould *et al.* 2002) is used to facilitate exchange of test and diagnostic data across applications developed by different vendors. DiagML is an XML-based format. A DiagML file provides sections for design, maintenance, test, and diagnostic data. The DiagML language focuses on traditional fault diagnosis, incorporating test strategies and parametric data for executing the tests.

While DiagML is very suitable for specifying test procedures across heterogeneous environments it does not provide support for modeling from first principles or constraint-based normative behavior of the system under test.

IDD: The European project "Integrated Design Process for onboard Diagnosis" (Struss *et al.* 2002) defines a number of XML-based model representations. Simulink numerical models provide structural and behavioral descriptions. These are converted to qualitative models which are later compiled to OBDD-like data structures to be used by an ATMS-based reasoner.

DTIF: The IEEE Digital Test Interchange Format specifies the information content and the data formats for the interchange of digital test program data between digital automated test program generators (DATPGs) and automatic test equipment (ATE) for board-level printed circuit assemblies. This information can be broadly grouped into data that defines the following: UUT Model, Stimulus and Response, Fault Dictionary, and Probe.

Although this is an IEEE standard, it is restricted to digital circuits and test-based diagnostic methodologies.

HSIF: The Hybrid Systems Interchange Format (Pinto *et al.* 2006) is probably the most advanced and most comprehensive format in existence today. It covers arguably the complete range of systems that one may want to diagnose. The main issue is extending this framework to make it more diagnosis-specific.

This framework is focused more on modeling systems, and less on interface specifications for implementing embedded systems.

OSA-CBM: The Open-Systems Architecture for Condition-Based Maintenance interchange format[1] was developed specifically for diagnosis and condition-based maintenance. In addition, it provides code-generators that can be used for creating interfaces for distributed sensors, actuators, and other inference modules.

In comparison to HSIF, this framework is higher-level, as it does not define semantics for equation types (e.g., dynamical equations), or of the transformations among equation types. This framework is focused more on interface specifications for implementing systems, and not on the specifics of modeling.

KIF: The Knowledge Interchange Format is a computer-oriented language for the interchange of knowledge among disparate programs. It has declarative semantics; it is logically comprehensive (i.e., it provides for the expression of arbitrary sentences in the first-order predicate calculus); it provides for the representation of knowledge about the representation of knowledge; it provides for the representation of nonmonotonic reasoning rules; and it provides for the definition of objects, functions, and relations.

XMLBIF: The Bayesian Network XML Interchange Format represents directed acyclic graphs that can be associated to conditional probability measures for discrete variables, with the possibility that decision and utility variables be present in the graph.

PSL: The Process Specification Language defines a vendor- and representation-neutral formalism for manufacturing processes. This may be important for representing life-cycle analysis issues, and not just one-time model specifications. Process data is used throughout the life cycle of a product, from early indications of manufacturing process flagged during design, through process planning, validation, production scheduling and control. In addition, the notion of process also underlies the entire manufacturing cycle, coordinating the workflow within engineering and shop floor manufacturing.

In building our proposal, we have considered a number of specialized formats for representing data structures of interest to MBD. These include BDDs, Petri Nets (Billington & others 2003), Netlists and decomposable NNFs (Darwiche 2001). Furthermore, many digital circuits are expressed in VHDL and Verilog, and we envision tools for translating subsets of these two languages.

Model Auto-Generation

Given the scarcity of MBD models and the cost of hand-constructing benchmark models, it is inevitable that automated model-generation techniques will be needed to provide appropriate model libraries. We now review two model generation approaches, given that the suite of tools associated with DIF will probably include auto-generation capabilities. The two approaches are: (a) a diagnosis generation tool using random-graph methods and model component libraries, and (b) circuit generation tools.

Diagnosis Model Auto-Generation: Recently, a diagnosis generation methodology based on graphical model generators has been proposed (Provan 2006). This work is aimed at auto-generating models for arbitrary systems, given a library of model components. The proposed methodology first generates a graph representing the system topology, and

[1] Cf. `http://osacbm.org/`.

then assigns system functionality using the component library, inserting a component for each node in the graph describing the system topology.

This approach is significant in that it can capture arbitrary systems. However, it is as accurate as (a) the topology generation mechanism and (b) the component library.

Random graph generators can effectively capture the gross topology of complex systems, but much work remains to more precisely capture detailed structure of particular domains. For example, the actual structure of the WWW is known to differ from the predictions of random graph models (Donato *et al.* 2004). In contrast, the practical applications and validity of the circuit-synthesis methods are more heavily-researched than the applications and validity of the random-graph generation approach; as a consequence, the models that a circuit-synthesis method generates are provably closer to the real-world targets (circuits) than are the models generated by random-graph generators are to their real-world targets, such as the WWW (Donato *et al.* 2004). However, many aspects of the circuit-generation algorithms are so particular to the precise architectures of circuits that they are not generalizable to other domains.

Circuit Benchmark Auto-Generation: A second group of related work addresses automatic benchmark circuit generation for improving the design of programmable logic architectures. A considerable literature exists for auto-generating circuits, including (Stroobandt, Verplaetse, & van Campenhout 1999; Kundarewich & Rose 2003; Holland & Hauck 2006).

Benchmark circuit auto-generation originally was based on applying a circuit generation rule, called Rent's rule (Landman & Russo 1971), but has since expanded to include other methods, e.g., as surveyed in (Adya *et al.* 2003).

Most automatic circuit generation methods are based on one of two methods, which we call equivalence-class and Rent-based methods. The equivalence-class methods (Ghosh & Brglez 1999) are based on perturbing a *seed circuit* to generate a circuit with similar overall structure but different local connectivity. The Rent-based methods use a power-law methodology, called Rent's rule, to generate circuits (Christie & Stroobandt 2000).

In the following, we examine the combinational circuit generation process, since this process has some properties that are potentially generalizable to any system model; sequential circuit generation addresses issues that are restricted to a *specific* class of temporal feedback systems with distinguished clock inputs, features that are not present in many other domains. Because of its greater generality, we focus on the Rent-based combinational circuit methods.

Rent's rule (Landman & Russo 1971), was originally derived empirically, but has since been given mathematical underpinnings. Rent's rule describes the relationship between the number of external signal connections to a logic block (called the number of "pins") and the number of logic gates in the logic block.

Rent's rule is given by $T = tn^{\xi}$, where (a) T is the number of input/output pins,[2] (b) n is the number of gates, (c) and the (internal) Rent exponent $0 \leq \xi \leq 1$ represents the level of placement optimization within a statistically homogeneous circuit, which is characterized by an interconnection topology with an average node degree t (or in engineering terms, t terminals per gate). From an engineering perspective, $\xi = 1$ corresponds to no placement optimization, i.e., the circuit is interpreted as a random gate arrangement. In actual circuits, the parameter ξ is dependent on circuit-topology: microprocessors, gate-arrays, and high-speed computers are characterized by Rent exponents of $\xi = 0.45, 0.5$, and 0.63, respectively (Christie & Stroobandt 2000).

Several tools have been developed to generate benchmark circuits based on Rent's rule and other approaches. Examples of such tools are CIRC and GEN (Kundarewich & Rose 2003). These tools can be integrated within the diagnostic model-generation framework described in this article.

Circuit generation algorithms have proven very useful for applications like FPGA design; however, they have two drawbacks when considered from the diagnostic model-generation viewpoint. First, they are restricted to a specific domain, and focus on *topology optimization*, rather than on the issues of fault isolation that are relevant to diagnosis benchmarks. As a consequence, these circuit generation algorithms have parameters that are pre-tuned to particular circuit classes; in contrast, a generic model generator must have parameters that can be assigned to generate models to approximate particular domains. It is important to note that any such parameters are domain-dependent, and need to be supplied by domain experts; in the absence of good domain parameters, the generated models will approximate real models with reasonable accuracy, the quality of which can be significantly improved with the use of precise parameters. Second, the circuit generation algorithms must be extended to incorporate a functional description that describes both normal and anomalous system behaviors.

Diagnosis Interchange Format

The DIF supports representation of models, observation vectors and diagnoses. Modeling semantics is a topic of research, and it is unlikely that our proposal would accommodate all the different approaches, ranging from hybrid systems with continuous time and state to static, discrete systems. In designing DIF, our main goals have been simplicity, translatability from existing implementation formats and extensibility.

Syntax and Semantics of DIF

As the standards described in this paper do not imply voluminous data according to current computing standards, and all the formats expose an ample amount of structure, our benchmark is encoded using the eXtensible Markup Language (XML) (Bray *et al.* 2006). The latter choice greatly simplifies the syntactical validation and representation, and

[2] In graph-theoretic terms, if we represent component i using a node in a topology graph, then the degree k_i of component i corresponds to the set of terminals of component i in the circuit-generation domain.

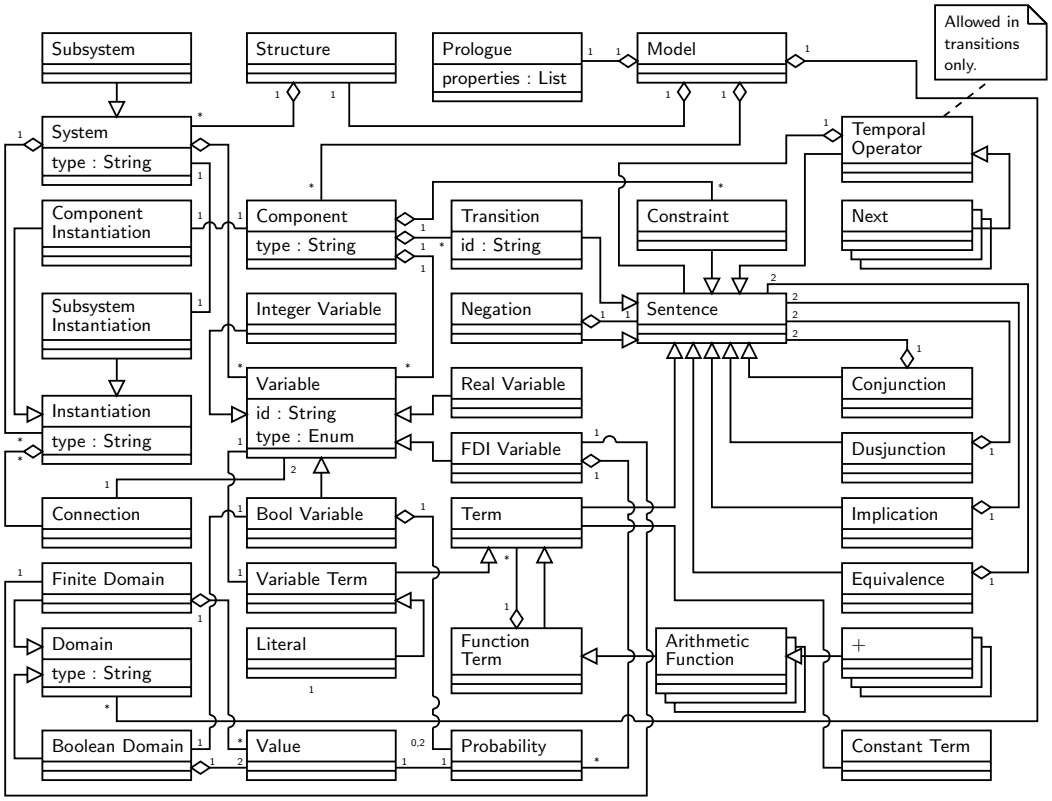


Figure 1: A visual representation of the DIF 1.0 model syntax (function classification, individual functions and some of the variable attributes are omitted).

allows the user to borrow from the vast amount of XML tooling. The DIF XML schema is visualized in Figure 1.

A DIF model has four sections: prologue, domains, structure, and components. The prologue describes the main characteristics of the model, i.e., the types of the constraints, fault-modeling, etc. The structure displays the model hierarchy that is essential for many of the MBD algorithms existing today. The domain description specifies symbolic values for all the Finite Domain Integer (FDI) variables (Booleans are treated as a special case of many-valued logic). Finally, for each component a set of constraints and transitions are specified. In particular, DIF supports constraints ranging from logic to differential equations.

DIF supports any First-Order Logic (FOL) sentence as a constraint, hence it provides for a large range of modeling techniques. The variables in a component or a subsystem, in addition to being Boolean or FDI can be in the real or (infinite) integer domains. For the latter two variable types, it is not necessary to specify domains. The framework is suitable for both abductive and consistency-based diagnosis.

A portion of the DIF schema is shown in Figure 2 and its full specification is available for download from `http://fdir.org/dif/`. Next, we provide some examples of how DIF can represent several classes of model.

A Combinatorial Circuit Example: Expressing structure is important, both from the algorithmic and modeling perspective. A DIF model typically specifies a number of hierarchical systems and a set of components. A system, then, can *instantiate* an arbitrary number of subsystems and components. Each system also defines a set of variables and how these variables are mapped into the systems and components it references. The use of hierarchy is best illustrated with the circuit shown in Figure 3.

The circuit is a full adder, each gate of which is allowed to fail without specifying faulty behavior. It consists of two half adders and an OR gate. The XML element in Figure 4 represents the structure in DIF (the variable mappings are omitted from the example for brevity). The top level system instantiates two copies of the half adder subsystem and an OR logic gate. The half adder uses an XOR and an AND gate. Note, that in the actual model the top-level system has some internal variables (f, p, and q) representing the wire connections.

The structure of a model is, essentially, a rooted, edge-labeled multidigraph $G = \langle V, E \rangle$, where the set of nodes V consists of all systems and components and there is an edge in E for each instantiation. One of the nodes is distinguished as a root (top-level) system. Constructing an algorithm that converts the hierarchical representation into a "flat" one is straightforward, by recursively merging the nodes of G.

A component model is given as a set of multi-valued propositional **Wff** over a set of variables V. A multi-valued variable $v_i \in V$ takes a value from a finite domain, which is

```
                ⋮
<xs:element name="model">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="prologue"/>
      <xs:element ref="domains" minOccurs="0"/>
                ⋮
<xs:element name="domains">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="domain">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="value"
                        type="xs:string"/>
          </xs:sequence>
                ⋮
<xs:complexType name="sentence">
  <xs:group ref="sentence"/>
</xs:complexType>
<xs:complexType name="transition">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="constraint">
      <xs:complexType>
        <xs:group ref="transition"/>
                ⋮
<xs:group name="sentence">
  <xs:choice>
    <xs:element name="equiv">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="sentence"/>
          <xs:group ref="sentence"/>
        </xs:sequence></xs:complexType>
                ⋮
```

Figure 2: Part of the DIF XML schema.

a set of symbols $D_i = \{s_1, s_2, \ldots, s_m\}$ with a 1-to-1 mapping to $\mathbb{Z}^+$. All the domains of FDI variables have to be explicitly specified in the second section of a DIF model. In the combinatorial circuit used for illustration, all variables are in the Boolean domain and the DIF definition of the latter is shown in Figure 5.

A positive multi-valued literal l_j^+ is a Boolean function $l_j^+ \equiv (v_i = d_k)$, where $v_i \in V$ and $d_k \in D_i$. A negative multi-valued literal l_j^- is defined in a similar fashion. A multi-valued propositional **Wff**, then, is a formula over the multi-valued literals $l_1, l_2, \ldots, l_n$, and the standard Boolean connectives: $\neg$ (negation), $\Leftrightarrow$ (equivalence), $\Rightarrow$ (implication), $\wedge$ (conjunction), and $\vee$ (disjunction).

Figure 6 shows the DIF specification of an XOR gate, which is a part of the sample full adder circuit introduced above. The component defines four variables: h, o, i_1, and i_2, of which h is assumable. The single constraint specifies the propositional **Wff** $h \Rightarrow (o \Leftrightarrow (i_1 \Leftrightarrow \neg i_2))$, the interpretation of which stipulates that the component health variable h is true iff the output o is true only when the values of i_1 and i_2 are different. The model of the component is weak-fault, i.e., if all the n components in a model are constrained by expressions in the form $h_i \Rightarrow F_i$, $1 \leq i \leq n$, where h_i is an assumable and does not appear in any of the propositional **Wff** F_j, $1 \leq i \leq n$, then the Minimal Diagnosis Hypothesis (de Kleer, Mackworth, & Reiter 1992) holds.

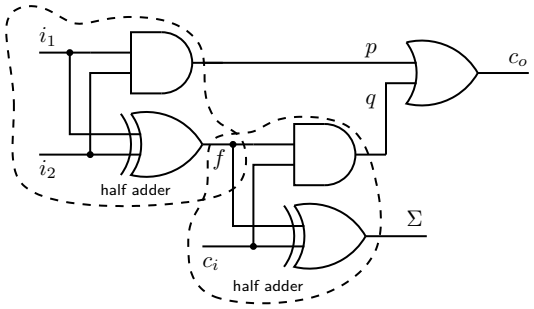


Figure 3: A full adder circuit.

```
<model>
    ⋮
  <structure>
    <system type="fullAdder">
      <subsysInst type="halfAdder" id="ha1" />
      <subsysInst type="halfAdder" id="ha2" />
      <compInstance type="orGate" id="orGate" />
    </system>
    <subsystem type="halfAdder">
      <compInst type="xorGate" id="xorGate" />
      <compInst type="andGate" id="andGate" />
    </subsystem>
  </structure>
```

Figure 4: Structure describing element of a full adder circuit.

Reasoning about time and state is central to model-based reasoning. Our discussion continues with some ways to represent dynamic characteristics of systems in DIF.

A Discrete Event System Model: DIF allows a component to define temporal constraints. The only assumption is that a diagnostic reasoner would maintain discrete time, and at every time step, it would copy all the variables and all the constraints for the current time instance.

A temporal constraint is a sentence in FOL that has variables from two instantiations of the system description in

```
<domains>
  <booleanDomain type="bool">
    <value default="true">true</value>
    <value>false</value>
  </booleanDomain>
</domains>
```

Figure 5: A DIF specification of the Boolean domain.

```
<component type="xorGate">
  <var domain="bool" id="h" type="health"/>
  <var domain="bool" id="o"/>
  <var domain="bool" id="i1"/>
  <var domain="bool" id="i2"/>
  <constraint>
    <imply><lit id="h"/>
      <equiv><lit id="o"/>
        <equiv><lit id="i1"/>
          <not><lit id="i2"/></not>
        </equiv>
      </equiv>
    </imply>
  </constraint>
</component>
```

Figure 6: A weak-fault model of an XOR logic gate.

time. The only difference between a temporal constraint and a combinatorial constraint is that a temporal constraint allows the use of "temporal operators".

An example of such a temporal operator is $\bigcirc$ (next) (the others include $\Box$ (globally), $\Diamond$ (eventually)) with semantics similar to the one in (Manna & Pnueli 1992). Note that $\bigcirc$ can only appear in front of a variable term, in which case $\neg \bigcirc x$ is equivalent to $\bigcirc \neg x$.

Our next example clarifies the DIF temporal semantics by discussing a model of a resettable pneumatic valve. The transition diagram of this valve is shown in Figure 7.

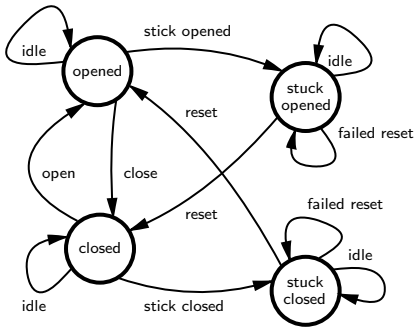


Figure 7: A transition diagram of a resettable valve.

The XML element shown in Figure 8 represents one of the possible transitions from Figure 7. It says that, given a positive assignment to the "reset" variable in the current instance, and if the valve is stuck open, it will change its state to closed when time progresses.

It is not possible to show all transitions of the valve in DIF as the full model specifies a transition for every edge of the graph shown in Figure 7. These constraints, however, are very similar to the one we have discussed in this section.

Before we continue with an example having an ODE for a constraint, it is worth noting that transitions are nothing more than set of constraints, having a name and being applied by the reasoners progressively over time.

```
<transition id="reset">
  <constraint><lit id="c">reset</lit></constraint>
  <constraint>
    <imply><lit id="s">stuckOpened</lit>
      <next><lit id="s">closed</lit></next>
    </imply>
  </constraint>
  <constraint>
    <imply><lit id="s">stuckClosed</lit>
      <next><lit id="s">opened</lit></next>
    </imply>
  </constraint>
</transition>
```

Figure 8: Valve transition from state "stuck closed" to "open" upon reset.

A Continuous System: Consider a numeric model of the primitive water clock, shown in Figure 9. The water level h (which was used in ancient times for approximating the time of the day) at time t is the solution of the ODE specified next to the figure. It is possible to build a fault model that specifies that the component is healthy if the value of h predicted by the numerical solution of the ODE, $\hat{h}$, is within a certain threshold δ, i.e., $|h - \hat{h}| < \delta$.

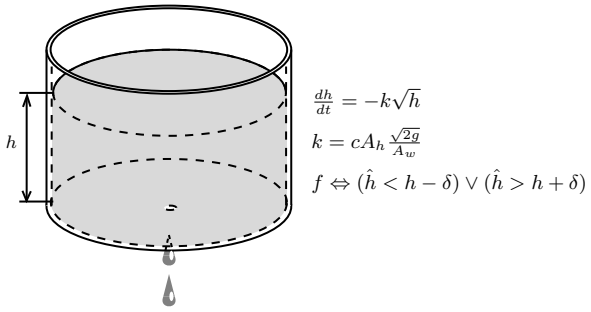


Figure 9: A set of hybrid constraints of a water clock fault model.

The rest of the parameters in Figure 9 are as follows. A_w and A_h are the cross-sectional areas of the water and the hole, respectively, and c is a friction constant. The acceleration due to gravity is denoted as g. The full model uses the Boolean fault variable f and an observable variable $\hat{h}$ in the real domain.

The DIF constraint shown in Figure 10 specifies an ODE, with t as the independent variable and both t and h in the continuous domain (both have type "real" in the variable declaration section of the full water clock model).

Having discussed DIF in specifying some models, we can continue with the remaining two data structures which are part of an MBD problem: observations and diagnoses.

Representation of Observations: In addition to models, an MBD format should specify syntax and semantics for observation vectors (sensor data). The semantics of an observation vector in DIF is illustrated in Figure 11. An obser-

```
<equiv>
  <fder><var id="h" /><var id="t" /></fder>
  <uminus>
    <prod>
      <var id="k" /><sqrt><var id="h" /></sqrt>
    </prod>
  </uminus>
</equiv>
```

Figure 10: An ODE constraint for the water clock shown in Figure 9.

vation vector in DIF is always a *conjunction* of variable assignments.

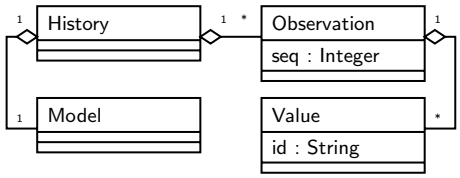


Figure 11: A visual representation of the DIF 1.0 observations syntax.

Figure 12 shows the DIF representation of a sample observation vector for the full-adder from Figure 3 (representing the propositional **Wff** $\mathrm{OBS}_2 = i_1 \wedge i_2 \wedge c_i \wedge \neg\Sigma \wedge c_o$). An observation refers to a specific instant in time. For the main MBD problem, a reasoner is supplied with a model in DIF and a sequence of observations in time, and it computes diagnoses, the format of which will be described later.

```
<obs seq="2">
  <lit id="i1"/><lit id="i2"/><lit id="ci"/>
  <not><lit id="sum"/></not><lit id="carry"/>
</obs>
```

Figure 12: An observation of the full adder from Figure 3.

As the problem of finding a kernel diagnosis is known to be Π_2^{P}-complete, most MBD implementations employ a *heuristic* based on the minimum number of failing components, or smallest probability failure mass (the DIF model language has a straightforward way for assigning probabilities to variable values). Hence the goal of an MBD benchmark is to provide an observation that leads to a *kernel diagnosis of minimum cardinality having the maximum number of failing components*. The latter problem is a topic on its own with many applications in MBR.

Representation of Diagnoses: The last part of our specification concerns the diagnoses as computed by MBD implementations. As both the observation and the diagnoses are sets of variable assignments, their formats are very similar. The class diagram for the DIF diagnosis representation is shown in Figure 13.

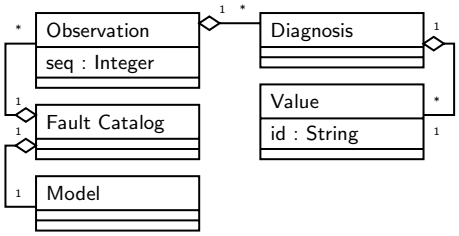


Figure 13: A visual representation of the DIF 1.0 diagnoses syntax.

A diagnosis file specifies zero or more diagnoses for some or all of the time instances at which observations have been performed. The observation of the full-adder (cf. Figure 3) from Figure 12 has been performed at time instance 2 according to its *seq* attribute. All kernel diagnoses of this observation are shown in Figure 14.

```
<obs seq="2">
  <diagnosis>
    <not><lit id="ha1.xorGate.h"/></not>
  </diagnosis>
  <diagnosis>
    <not><lit id="ha2.xorGate.h"/></not>
  </diagnosis>
</obs>
```

Figure 14: A diagnosis of the full adder from Figure 3, given the observation from Figure 12.

We use the dotted notation in the literal identifiers to specify the subsystems in which the component resides. The two possible kernel diagnoses shown in Figure 14 are each of the two XOR gates of the full-adder being faulty.

MBD Benchmark

Advances in the MBD algorithms will quickly obsolete a static set of benchmark problems, hence we see the maintenance of a benchmark suite as an ongoing effort. Providing a benchmark set is also technologically challenging. We have compiled an initial set of benchmark problems and published them on `http://fdir.org/dif/`.

Our initial benchmark proposal consists of three categories: (1) combinatorial circuits, (2) random models, and (3) real-world systems. In addition to the models, we have provided observation data as discussed in this paper. In some cases diagnostic results are included as well.

The ISCAS-85 set of combinatorial circuits (Brglez & Fujiwara 1985) has been used as the *de facto* benchmark in MBD and we have translated it to DIF. In order to facilitate hierarchical solvers, in addition to the traditional flat representation, we have provided DIF versions of the reverse engineered high-level ISCAS-85 circuits (Hansen, Yalcin, & Hayes 1999).

For the random diagnosis problems, we have used a model generator as described in the beginning of this paper. The

random problems have different numbers of components, observable variables and connectivity.

Finally, we have anonymized some real-world models of practical significance and added them to the benchmark suite. A full description of the benchmark problems is not possible due to the limitations in the paper length and is available from the benchmark web site.

Conclusion

This article have proposed a Diagnosis Interchange Format (DIF) intended to facilitate (a) exchange of models, component libraries, sensor data (observation vectors) and diagnoses in MBD, and (b) empirical comparative studies of the performance of existing and future MBD algorithms.

In the article we have summarized the syntax and the semantics of DIF, as well as the principles underlying its design. We argue that DIF is a compact representation for representing diagnosis models. By using inheritance and specialization, we showed how a very broad modeling language like DIF can be specialized to represent explicit models such as digital circuits. This specialization decreases the modeling complexity and allows modelers to use DIF for fault-diagnosis of Boolean circuits, for example, while not being burdened with the language's expressiveness outside the domain of propositional logic.

While DIF may be compact for representing a wide range of systems, MBD employs a variety of representations allowing trade-offs in time, space, and off-line time (i.e., knowledge compilation approaches). A future extension of this standard would benefit from supporting compact compiled representations like NNF (Negation Normal Forms), OBDD (Ordered Binary Decision Diagrams) and others.

Acknowledgments

This work has been supported by STW grant DES.7015 and SFI grant 04/IN3/I524.

References

Adya, S. N.; Yildiz, M. C.; Markov, I. L.; Villarrubia, P. G.; Parakh, P. N.; and Madden, P. H. 2003. Benchmarking for large-scale placement and beyond. In *Proc. ISPD'03*, 95–103.

Benazera, E.; Travé-Massuyès, L.; and Dague, P. 2002. State tracking of uncertain hybrid concurrent systems. In *Proc. DX'02*, 106–114.

Billington, J., et al. 2003. The Petri net markup language: Concepts, technology, and tools.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2006. Extensible markup language (XML) 1.0. Technical Report REC-xml-20060816, W3C.

Brglez, F., and Fujiwara, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. ISCAS'85*, 695–698.

Carlson, B., and Gupta, V. 1998. Hybrid cc with interval constraints. In *Proc. HSCC'98*, 80–95.

Christie, P., and Stroobandt, D. 2000. The interpretation and application of Rent's rule. *IEEE Trans. Very Large Scale Integr. Syst.* 8(6):639–648.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48(4):608–647.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3):197–222.

Donato, D.; Laura, L.; Leonardi, S.; and Millozzi, S. 2004. Simulating the webgraph: A comparative analysis of models. *Computing in Science and Engineering* 6(6):84–89.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM* 42(1):3–42.

Ghosh, D., and Brglez, F. 1999. Equivalence classes of circuit mutants for experimental design. In *Proc. ISCAS'99*, 432–435.

Gould, E.; Hartop, D.; Lee, E.; Neag, I. A.; and Wilson, M. 2002. Diagml - an interoperability platform for test and diagnostics software. In *Proc. of IEEE AUTOTESTCON-02*, 597 – 607.

Hansen, M.; Yalcin, H.; and Hayes, J. 1999. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test* 16(3):72–80.

Holland, M., and Hauck, S. 2006. Improving performance and robustness of domain-specific CPLDs. In *Proc. FPGA'06*, 50–59.

Kundarewich, P. D., and Rose, J. 2003. Synthetic circuit generation using clustering and iteration. In *Proc. FPGA'03*, 245–245.

Landman, B. S., and Russo, R. L. 1971. On pin versus block relationship for partitions of logic circuits. *IEEE Trans. Computers* 20(6):1469–1479.

Manna, Z., and Pnueli, A. 1992. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag.

Pietersma, J.; Feldman, A.; and van Gemund, A. 2006. Modeling and compilation aspects of fault diagnosis complexity. In *Proc. of IEEE AUTOTESTCON-06*.

Pinto, A.; Carloni, L. P.; Passerone, R.; and Sangiovanni-Vincentelli, A. 2006. Interchange formats for hybrid systems: Abstract semantics. In *Proc. Hybrid Systems: Computation and Control*, 491–506.

Provan, G., and Wang, J. 2007. Automated benchmark model generators for model-based diagnostic inference. In *Proc. IJCAI'07*, 513–518.

Provan, G. 2006. Automated benchmark model generators for model-based diagnostic inference. In *Proc. DX'06*, 99–104.

Sheppard, J. W., and Orlidge, L. A. 1997. Artificial intelligence exchange and service tie to all test environments (AI-ESTATE) - a new standard for system diagnostics. In *Proc. of the IEEE International Test Conference*, 1020–1029.

Stroobandt, D.; Verplaetse, P.; and van Campenhout, J. 1999. Towards synthetic benchmark circuits for evaluating timing-driven cad tools. In *Proc. ISPD'99*, 60–66.

Struss, P.; Rehfus, B.; Brignolo, R.; Cascio, F.; Console, L.; Dague, P.; Dubois, P.; Dressler, P.; and Millet, D. 2002. Model-based tools for the integration of design and diagnosis into a common process - a project report. In *Proc. DX'02*.

Williams, B., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Proc. AAAI'96*, 971–978.

Automated Debugging and Repair of Utility Constraints in Recommender Knowledge Bases

A. Felfernig[1], G. Friedrich[1], E. Teppan[1]

[1]University Klagenfurt, Intelligent Systems and Business Informatics,
Universitaetsstrasse 65-67, A-9020 Klagenfurt, Austria
{*felfernig*, *friedrich*, *teppan*}*@ifit.uni-klu.ac.at*
phone: +43/463/2700/3705; fax: +43/463/2700/3799

Abstract

Recommendation problems related to complex products and services (e.g., computers, financial services, digital cameras) are a thriving application area for knowledge-based technologies. Automated debugging support for recommender knowledge bases is a necessary prerequisite for supporting effective development and maintenance processes. In this paper we present an approach to the automated debugging of constraint sets which specify the way utilities of product alternatives are determined. In many cases, such constraint sets do not calculate the rankings expected by marketing and sales experts and therefore have to be adapted. By applying Model-Based Diagnosis, we show how faulty utility constraints can be automatically identified and adapted by exploiting examples for intended product rankings provided by domain experts or derived from existing customer interaction logs. The presented debugging technologies have been implemented for a commercial recommender development environment.

Introduction

Recommender applications improve the accessibility of large product assortments. There are three basic recommendation approaches. One of the most frequently used one is *Collaborative Filtering* (Herlocker et al. 2004) which is an implementation of word-of-mouth promotion where buying decisions are influenced by the opinions of friends and benchmarking reports. *Content-based Filtering* (Pazzani 1999) is an information filtering approach exploiting item features a user has liked in the past with the goal of recommending new items. Content-based filtering recommends all items based on purchase information available from the *current* customer. Both approaches are based on long-term user models and do not exploit deep knowledge about the product domain. In contrast, knowledge-based recommender technologies exploit deep knowledge about the product domain (Burke 2000, Felfernig and Kiener 2005) in order to determine solutions fitting to the wishes and needs of (online) customers. Compared to customers purchasing books or movies, customers purchasing complex products such as computers, financial services or digital cameras are much more in the need of intelligent interaction mechanisms supporting the calculation and explanation of appropriate solutions. Therefore, an explicit representation of product, marketing, and sales knowledge is needed which makes it possible (a) to derive recommendations which comply with existing marketing and sales strategies and suit the wishes of a customer, (b) to explain those recommendations, and (c) to propose repair actions for inconsistent requirements.

Product alternatives calculated for a set of customer requirements are typically ranked according to their utility for the customer. Such rankings are calculated in order to exploit serial positioning effects which cause consumers to preferably select items at the beginning of a list (Gershberg and Shimamura 1994, Coulter and Coulter 2005). For the determination of personalized product rankings we have integrated the concepts of *Multi Attribute Utility Theory* (*MAUT*) (Winterfeldt and Edwards 1986) into our recommender development environment (Felfernig and Kiener 2005). A precondition for applying MAUT is the definition of *utility constraints* (scoring rules). These constraints specify to which extent different customer requirements contribute to *interest dimensions*, e.g., if a customer requires a digital camera with a low resolution, we can infer a high interest in recommendations with *low costs*. Furthermore, utility constraints define the extent to which product alternatives contribute to the defined interest dimensions, e.g., a digital camera with a low resolution fits to customers interested in low-cost cameras and therefore provides a high value for the interest dimension *economy*.

One of the major challenges imposed by the application of MAUT is the maintenance of utility constraints (Felfernig and Kiener 2005). The relationship between customer requirements, products, and corresponding interest dimensions has to be explicitly specified by knowledge engineers. Such constraints have to reflect marketing and sales strategies which specify rules such as *if* $digicam_a$ *is part of the recommendation result and* $digicam_b$ *is part of the result as well, then the utility of* $digicam_a$ *should be higher than the one of* $digicam_b$. Experiences from commercial projects (Felfernig and Kiener 2005) clearly point out the need for additional knowledge acquisition support in this context. The maintenance of utility constraint sets is a time-consuming and error-prone task since in many cases those constraints are strongly

interdependent. Therefore, our goal was to develop techniques which support knowledge engineers in the identification of faulty utility constraints. In this paper we present debugging concepts which automatically identify sources of inconsistencies in utility constraint sets and propose the corresponding repair actions. By exploiting the concepts of Model-based Diagnosis (MBD) (Reiter 1987, de Kleer et al. 1992) we are able to automatically identify and repair minimal sets of faulty constraints. The approach presented in this paper has been developed for a commercial recommender development environment (Felfernig and Kiener 2005) and is in the line of previous work related to the debugging of recommender user interfaces (Felfernig et al. 2006).

The remainder of the paper is organized as follows. The following section presents utility constraints which are used as working example throughout the paper. Section *CSP-based Representation of Utilities* sketches our approach to transform a given set of utility definitions into a corresponding Constraint Satisfaction Problem (CSP) representation which is the basis for applying MBD. Section *Debugging Utility Constraints* presents our approach to the automated identification and repair of faulty constraints in MAUT constraint sets.

Example: Utility Definitions

We now present a simplified set of utility constraints (scoring rules) which is used as working example throughout the paper. The basic elements of MAUT are interest dimensions such as *quality* or *economy* describing interest focuses of a customer, e.g., *quality* denotes the output quality of a digital camera, *economy* is related to financial aspects such as price, maintenance, etc. The degree to which a customer is interested in such dimensions can be directly derived from the requirements articulated within the scope of a recommendation session (see Tables 1-2). A customer who is interested in *6MPIX* cameras (*minres=6MPIX*) has a higher interest in *quality* than a customer who is interested in purchasing a *5MPIX* camera (*minres=5MPIX*). Similarly, customers interested in low prices (*maxprice>=200* and *maxprice<=300*) have a lower interest in *quality* than those with less restrictions related to the price (*maxprice>=301* and *maxprice<=1000*).

minres	quality	eco
5MPIX	5	9
6MPIX	8	5

Table 1: Scoring rules for customer property *minres*.

maxprice	quality	eco
200-300	3	7
301-1000	9	2

Table 2: Scoring rules for customer property *maxprice*.

For the purposes of our example, we use the set of customer requirements specified in Table 3.[1]

customer	minres	maxprice
$customer_1$	6MPIX	1000
$customer_2$	6MPIX	300

Table 3: Customer requirements.

Using the Tables 1-3 we can determine a distribution for the interest dimensions of our example customers, i.e., to which extent those customers are interested in the specified interest dimensions *quality* and *economy*. Both customers require a *6MPIX* camera which contributes an importance of 8 to the interest dimension *quality* and an importance of 5 to the interest dimension *economy* (Table 4 depicts the complete set of evaluations).

customer	quality	eco
$customer_1$	8+9=**17**	5+2=**7**
$customer_2$	8+3=**11**	5+7=**12**

Table 4: Example customer preferences.

On the basis of such customer preferences, we are able to evaluate which of a given set of alternative products best suits the customer's wishes and needs. For the purpose of our simple example, we introduce the following simplified assortment ($digicam_{a,b}$) depicted in Table 5.

name	company	resolution	price
$digicam_a$	A	6MPIX	290
$digicam_b$	B	6MPIX	310

Table 5: Example set of digital cameras.

After having specified our example product assortment, we have to define the utility of those products w.r.t. the specified set of interest dimensions. We define a dependence between product attribute values and the interest dimensions *quality* and *economy*, e.g., cameras with higher resolutions support higher output *quality* (see Table 6), cameras with a lower price have a better evaluation w.r.t. the dimension *economy* (see Table 7), and cameras provided by company *A* have a higher *quality* than cameras offered by company *B* – vice-versa the *economy* of cameras from company *B* is better than the *economy* of cameras from company *A* (see Table 8). Note that in real-world applications, product properties and customer properties can differ significantly, whereas in our example the single difference is the product attribute *company*.

resolution	quality	eco
6MPIX	9	4

Table 6: Scoring rules for product attribute *resolution*.

[1] For the reasons of clearness and simplicity we restrict our working example to two digital cameras (both equipped with a *6MPIX* resolution).

price	quality	eco
200-300	4	7
301-1000	8	2

Table 7: Scoring rules for product attribute *price*.

company	quality	eco
A	8	7
B	6	9

Table 8: Scoring rules for product attribute *company*.

Using the Tables 4-8 we can determine the extent to which our cameras contribute to the interest dimensions *quality* and *economy* (see Table 9): $digicam_a$ has a lower *quality* than $digicam_b$ but a higher *economy* value. Vice-versa, $digicam_b$ has a lower value of *economy* than $digicam_a$.

name	quality	eco
$digicam_a$	9+4+8=**21**	4+7+7=**18**
$digicam_b$	9+8+6=**23**	4+2+9=**15**

Table 9: Utilities for products w.r.t. interest dimensions.

On the basis of the identified product utilities, we can determine the customer-specific utility of each product (for details see Table 10). This utility of a product can be determined by using the formula

$$utility(x) = \sum_{i=1}^{n} in_i con_i(x)$$

where *utility*(x) specifies the overall utility of a product x for a specific customer. The overall utility is defined as sum over the customer's interest in dimension i (in_i) (see Table 4) times the contribution of product x to interest dimension i (con_i) (see Table 9).

In our example, $digicam_b$ has a higher utility for $customer_1$, whereas $digicam_a$ has a higher value for $customer_2$.

customer	product	quality	eco	utility
$customer_1$	$digicam_a$	21*17	18*7	483
	$digicam_b$	23*17	15*7	**496**
$customer_2$	**$digicam_a$**	21*11	18*12	**447**
	$digicam_b$	23*11	15*12	433

Table 10: Utilities of products for customers.

Our goal is it to identify a minimal set of repair actions for utility constraints which are consistent with examples for product rankings provided by online customers (representative interaction sequences) or marketing and sales experts. Such examples for the intended behavior of utility constraints have the following structure (see Table 11):

example	minres	maxprice	ranking
e_1	6MPIX	1000	utility($digcam_a$) < utility($digicam_b$)
e_2	6MPIX	300	utility($digcam_a$) < utility($digicam_b$)

Table 11: Examples for product rankings.

Example e_1 denotes the fact that in all interaction sequences with the recommender application where customers require a digital camera with at least *6MPIX* and a maximum price of *1000*, the utility of $digicam_a$ should be lower than the utility of $digicam_b$, i.e., $digicam_a$ should be ranked at a position in the result set which is located after the position of $digicam_b$. Furthermore, if a customer requires a digital camera with at least *6MPIX* and a maximum price of *300*, then the utility of $digicam_a$ should be as well lower than the utility of $digicam_b$.

CSP-based Representation of Utilities

In order to be able to apply Model-based Diagnosis (Reiter 1987, de Kleer et al. 1992) to the identification and repair of faulty utility constraints, we transform a given set of MAUT utilities into a corresponding Constraint Satisfaction Problem (CSP). Model-based Diagnosis starts with the description of a system which is in our case the description of the intended behaviour of a given set of utility constraints. This intended behaviour is specified by a set of examples describing the intended ordering of products in a result set depending on a set of customer requirements (see Table 11). If the actual system behavior (rankings calculated by the utility constraints) conflicts with its intended behaviour (rankings described by examples), the diagnosis task is to determine those components (utility constraints) which, when assumed to functioning abnormally, explain the discrepancy between actual and intended behaviour.

Following the definitions of Tables 1-2, we introduce the following set of utility constraints related to the expected resolution and accepted price of a digital camera, e.g., constraint c_1 denotes the fact that for customers requiring a digital camera with at least *6MPIX* the dimension *quality* is important, whereas *economy* aspects play a less important role (c_2). The constraints $\{c_1,c_2\}$ represent the utility definitions of Table 1 (the first entry has not been taken into account), $\{c_3,c_4,c_5,c_6\}$ represent the definitions of Table 2.

c_1: $quality_{(minres\_6MPIX)} = 8$
c_2: $eco_{(minres\_6MPIX)} = 5$
c_3: $quality_{(maxprice\_200\_300)} = 3$
c_4: $eco_{(maxprice\_200\_300)} = 7$
c_5: $quality_{(maxprice\_301\_1000)} = 9$
c_6: $eco_{(maxprice\_301\_1000)} = 2$

We denote each constraint defining such utility values as utility constraint $c_i \in C$. Since we are interested in a utility constraint set which is consistent with all the examples $e_i \in E$, we have to check the consistency of the given set of utility constraints with $\cup e_i$. This type of consistency check requires a representation where each example is described by a separate set of finite domain variables, e.g., the contribution to *quality* provided by the customer attribute *minres* in example e_1 is stored in the variable $quality_{(minres\_e1)}$. The following representation of examples can be directly applied by our diagnosis algorithm.

e_1: $quality_{(minres\_e1)} = quality_{(minres\_6MPIX)} \wedge$
$eco_{(minres\_e1)} = eco_{(minres\_6MPIX)} \wedge$
$quality_{(maxprice\_e1)} = quality_{(maxprice\_301\_1000)} \wedge$

$eco_{(maxprice\_e1)} = eco_{(maxprice\_301\_1000)} \wedge$
$utility_{(digicama\_e1)} < utility_{(digicamb\_e1)}$

e_2: $quality_{(minres\_e2)} = quality_{(minres\_6MPIX)} \wedge$
$eco_{(minres\_e2)} = eco_{(minres\_6MPIX)} \wedge$
$quality_{(maxprice\_e2)} = quality_{(maxprice\_200\_300)} \wedge$
$eco_{(maxprice\_e2)} = eco_{(maxprice\_200\_300)} \wedge$
$utility_{(digicama\_e2)} < utility_{(digicamb\_e2)}$

The overall customer interest (customer of e_1) in the dimension *quality* is stored in $quality_{(customer\_e1)}$. The value of this variable represents the sum over all defined contributions of customer requirements of example e_1 to the dimension *quality*. This approach is analogously applied to the dimension *economy* ($eco_{(customer\_e1)}$). We denote constraints summing up customer utilities as $s_{ij} \in S$. The following constraints implement the definitions of Table 4.

s_{11}: $quality_{(customer\_e1)} = quality_{(minres\_e1)} + quality_{(maxprice\_e1)}$
s_{12}: $eco_{(customer\_e1)} = eco_{(minres\_e1)} + eco_{(maxprice\_e1)}$
s_{21}: $quality_{(customer\_e2)} = quality_{(minres\_e2)} + quality_{(maxprice\_e2)}$
s_{22}: $eco_{(customer\_e2)} = eco_{(minres\_e2)} + eco_{(maxprice\_e2)}$

For each product part of our example assortment, we specify its contribution to the given interest dimensions, e.g., the price specified for product $digicam_a$ (290) defines an average interest in the dimension *quality*. In our CSP, we define this fact as $quality_{price(digicama)} = 4$. Analogously, we define the relationship between the interest dimension *economy* and price ($eco_{price(digicama)} = 7$). We denote each constraint defining a utility value for a certain product as utility constraint $p_j \in P$. The following constraints implement the definitions of Tables 6-8.

p_1: $quality_{price(digicama)}=4$ p_2: $eco_{price(digicama)}=7$
p_3: $quality_{price(digicamb)}=8$ p_4: $eco_{price(digicamb)}=2$
p_5: $quality_{resolution(digicama)}=9$ p_6: $eco_{resolution(digicama)}=4$
p_7: $quality_{resolution(digicamb)}=9$ p_8: $eco_{resolution(digicamb)}=4$
p_9: $quality_{company(digicama)}=8$ p_{10}: $eco_{company(digicama)}=7$
p_{11}: $quality_{company(digicamb)}=6$ p_{12}: $eco_{company(digicamb)}=9$

For each $c_i \in C$ (and each $p_i \in P$) we add a corresponding repair constraint cr_i (pr_i) which specifies possible repairs for c_i (p_i). The idea behind repair constraints is that if c_i (p_i) is identified as faulty element by the diagnosis component, we have to identify a consistent repair action for c_i (p_i). This repair should change the original c_i (p_i) as little as possible[2]. Therefore, we define an interval for the accepted changes for each $c_i \in C$ and each $p_i \in P$. Each of the following example repair constraints allows changes of the given evaluations by at most one unit. We denote $\cup cr_i \cup \cup pr_i$ as the set of repair constraints R.

cr_1: $quality_{(minres\_6MPIX)}$ IN [7,8,9]
cr_2: $eco_{(minres\_6MPIX)}$ IN [4,5,6]
cr_3: $quality_{(maxprice\_200\_300)}$ IN [2,3,4]
cr_4: $eco_{(maxprice\_200\_300)}$ IN [6,7,8]
cr_5: $quality_{(maxprice\_301\_1000)}$ IN [8,9,10]
cr_6: $eco_{(maxprice\_301\_1000)}$ IN [1,2,3]

pr_1: $quality_{price(digicama)}$ IN [3,4,5]
pr_2: $eco_{price(digicama)}$ IN [6,7,8]
pr_3: $quality_{price(digicamb)}$ IN [7,8,9]
pr_4: $eco_{price(digicamb)}$ IN [1,2,3]
pr_5: $quality_{resolution(digicama)}$ IN [8,9,10]
pr_6: $eco_{resolution(digicama)}$ IN [3,4,5]
pr_7: $quality_{resolution(digicamb)}$ IN [8,9,10]
pr_8: $eco_{resolution(digicamb)}$ IN [3,4,5]
pr_9: $quality_{company(digicama)}$ IN [7,8,9]
pr_{10}: $eco_{company(digicama)}$ IN [6,7,8]
pr_{11}: $quality_{company(digicamb)}$ IN [5,6,7]
pr_{12}: $eco_{company(digicamb)}$ IN [8,9,10]

The *quality* of a product is defined by the sum of contributions of the values of *price*, *resolution*, and *company*. *Economy* of a product is as well defined by the sum of related contributions of those attribute values. We denote each rule summing up product utility values as $s_i \in S$. The following constraints implement the definitions of Table 9.

s_1: $quality_{(digicama)} =$
$quality_{price(digicama)} +$
$quality_{resolution(digicama)} +$
$quality_{company(digicama)}$
s_2: $quality_{(digicamb)} =$
$quality_{price(digicamb)} +$
$quality_{resolution(digicamb)} +$
$quality_{company(digicamb)}$
s_3: $eco_{(digicama)} =$
$eco_{price(digicama)} +$
$eco_{resolution(digicama)} +$
$eco_{company(digicama)}$
s_4: $eco_{(digicamb)} =$
$eco_{price(digicamb)} +$
$eco_{resolution(digicamb)} +$
$eco_{company(digicamb)}$

The following constraints specify the calculation of product utilities, where $utility_{(x\_ei)}$ specifies the utility of product *x* in the context of example e_i (see Table 10).

s_{13}: $utility_{(digicama\_e1)} =$
$quality_{(digicama)} * quality_{(customer\_e1)} +$
$eco_{(digicama)} * eco_{(customer\_e1)}$
s_{14}: $utility_{(digicamb\_e1)} =$
$quality_{(digicamb)} * quality_{(customer\_e1)} +$
$eco_{(digicamb)} * eco_{(customer\_e1)}$
s_{23}: $utility_{(digicama\_e2)} =$
$quality_{(digicama)} * quality_{(customer\_e2)} +$
$eco_{(digicama)} * eco_{(customer\_e2)}$
s_{24}: $utility_{(digicamb\_e2)} =$
$quality_{(digicamb)} * quality_{(customer\_e2)} +$
$eco_{(digicamb)} * eco_{(customer\_e2)}$

All the mentioned constraints are constituting elements of a corresponding Constraint Satisfaction Problem which is defined by the tuple (V, D, Con), where V is a set of finite integer domain variables, D represents the variables' domain, and Con is a set of constraints consisting of exactly those constraints defined in (C, P, R, S, E).

[2] Note that changes of at most one unit are only introduced for this example, the range of possible changes in general is more flexible: in the current implementation it can be specified by knowledge engineers.

Debugging Utility Constraint Sets

A major challenge when implementing recommender applications is the identification of appropriate utility scores (constraints). Our approach to support associated knowledge engineering processes is to apply the concepts of MBD which allows us to identify minimal constraint subsets which have to be changed in order to better fit to the provided set of example rankings. In the sense of MBD, a faulty component (erroneous utility constraint) is expressed through an inconsistency between the utility constraint set and the specified examples (see the following definition of a *MAUTKB Diagnosis Problem* and a *MAUTKB Diagnosis*). A MAUTKB diagnosis indicates faulty constraints whose adaptation will restore consistency. We exploit examples which are provided by domain experts or online customers. Those examples specify the expected ranking of product recommendations. Depending on a set of customer requirements, the recommender should rank products in a way which is consistent with the rankings specified in the given set of examples (in our case $\{e_1, e_2\}$).

With regard to our example utility constraints, $\{e_1\} \cup C \cup P \cup R \cup S$ is consistent. Adding $\{e_2\}$ to $\{e_1\} \cup C \cup P \cup R \cup S$ triggers an inconsistency which can be restored by deleting $\{p_3, p_{12}\}$ from P. In this context, $\{p_3, p_{12}\}$ is a minimal diagnosis, i.e., a minimum set Δ of constraints which have to be deleted from $C \cup P \cup R$, s.t. $\{e_1, e_2\} \cup C \cup P \cup R - \Delta \cup S$ becomes consistent. Based on this approach to the identification of faulty utility constraints, we introduce a definition of a MAUTKB Diagnosis Problem and a corresponding MAUTKB Diagnosis.

MAUTKB Diagnosis Problem. A MAUTKB Diagnosis Problem (MAUT Knowledge Base Diagnosis Problem) is defined as a quintuple (C, P, R, S, E), where C is a set of customer utility constraints, P is a set of utility rules related to offered products, R is set of repair constraints, and S represents a set of summarization rules. Finally, E is a set of examples for the intended ordering of result sets. □

Based on this definition of a MAUTKB Diagnosis Problem, we now introduce the definition of a corresponding MAUTKB Diagnosis.

MAUTKB Diagnosis. A MAUTKB Diagnosis for (C, P, R, S, E) is a set $\Delta \subseteq C \cup P \cup R$, s.t. $(C \cup P \cup R) - \Delta \cup S \cup E$ is consistent. □

Note that a diagnosis Δ for (C, P, R, S, E) always exists under the reasonable assumption that (a) each single example is consistent, (b) the examples do not interfere with each other, and (c) $S \cup E$ is consistent, i.e., if all constraints in $C \cup P \cup R$ are contained in Δ, $S \cup E$ should support product rankings consistent with E (Felfernig et al. 2007). Note that in order to assure the existence of a diagnosis, we allow constraints $\in$ R to be diagnosable.

The calculation of diagnoses is based on the determination of minimal conflicts induced by the examples $e_i \in E$.

Conflict Set CS. A Conflict Set is defined as a minimal subset $CS = \{c_1, c_2, \ldots, c_k\} \subseteq C \cup P \cup R$ s.t. $CS \cup S \cup E$ inconsistent. CS is minimal iff *not*($\exists$ CS': CS' $\subset$ CS) □

Properties of Diagnosis Algorithm. For the calculation of diagnoses we have implemented an extended version of the algorithm proposed by (Reiter 1987). The labelling of the search tree is based on the labelling of the original HSDAG (Hitting Set Directed Acyclic Graph). A node n is labelled by a minimal conflict set CS(n). The set of edge labels from the root to node n is referred to as H(n). One minimal diagnosis Δ for our example is $\{p_3, p_{12}\}$, the corresponding repairs are, e.g., $\{p_3'$: $quality_{price(\text{digicamb})}=9$, p_{12}': $eco_{company(\text{digicamb})}=10\}$. Introducing these repairs makes $(C \cup P \cup R) - \{p_3, p_{12}\} \cup \{p_3', p_{12}'\} \cup S \cup E$ consistent.

The algorithm for calculating a set of minimal diagnoses for a given set of utility constraints is the following (Algorithm 1). In each step of the HSDAG construction, the algorithm checks the consistency of $(C \cup P \cup R) - \Delta \cup S \cup E$. In the case of an inconsistency, the theorem prover provides a minimal conflict set which is resolved by the expansion of the HSDAG, otherwise H(n) represents one (alternative) minimal diagnosis.

Algorithm 1 MAUTKB-Diagnosis (C, P, R, S, E)

(a) Generate a pruned HSDAG T for the collection of minimal conflict sets induced by constraints defined in $C \cup P \cup R$ in breadth-first manner (we generate diagnoses in order of their cardinality). With every theorem prover (TP) call at a node n of T the consistency of $(C \cup P \cup R - H(n) \cup S \cup E)$ is checked. If there exists an inconsistency, a minimal conflict set CS is returned, otherwise *ok* is returned. If $(C \cup P \cup R - H(n) \cup S \cup E)$ is consistent, a corresponding diagnosis H(n) is found.

(b) Return $\{H(n) \mid$ n is a node of T labelled with *ok*$\}$.

The algorithm has been implemented as part of our knowledge-based recommender environment (Felfernig and Kiener 2005). The calculation of minimal conflict sets for the construction of the HSDAG is based on the QUICKXPLAIN algorithm proposed in (Junker 2004). The algorithm needs $O(n*\log(k+1)+k^2)$ consistency checks to compute a minimal conflict of size *k* out of *n* constraints in $(C \cup P \cup R)$. The performance of Algorithm 1 (HSDAG construction including conflict detection) allows an application for interactive settings where knowledge engineers are developing and maintaining sets of utility constraints (Felfernig et al. 2007). Conform to the original HSDAG algorithm (Reiter 1987), our approach calculates for each tree level the complete set of possible diagnoses. In our implementation, we favourably present diagnoses which do not include any elements of R, i.e, only propose changes to the defined utility constraints in $C \cup P$ (scoring rules). Finally, the current implementation of our diagnosis algorithm exploits examples defined by knowledge engineers and marketing & sales experts. Future versions

will take into account the derivation of examples from existing interaction logs.

Empirical Results

The purpose of the performance evaluation was to measure the time needed by the diagnosis algorithm to find a minimal diagnosis which makes a MAUT constraint set consistent with given example orderings.

To this end, three MAUT-constraint sets (CS_1, CS_2, CS_3) of different complexity levels in terms of the number of variables and the number of constraints have been designed (see Table 12).

	Variables	Constraints
CS_1	48	36
CS_2	194	127
CS_3	238	166

Table 12: Complexity of MAUT constraint sets.

For every MAUT constraint set, test sets (example sets) (E_1, E_2, E_3) have been designed, which triggered an increasing number of conflicts. The MAUT constraint sets in combination with the example sets reflect representative repair situations occurring in commercial settings (Felfernig and Kiener, 2005). The time efforts shown in Table 13 represent the time needed for the calculation of the first diagnosis (interactive mode, where the knowledge engineer is confronted with one diagnosis at a given time). The tests have been carried out on a Pentium D machine with a clock rate of 2800 MHz and a RAM size of 2048 MByte. In typical settings, the analysis of a MAUT constraint set should take place about every two months. This makes the actual response times acceptable. However, improving the efficiency of the algorithm is in the focus of future work. We are currently working on the integration of genetic algorithms for improving the efficiency of solution search (Back and Schwefel, 1993).

		E_1	E_2	E_3
CS_1	# Conflicts	1	2	11
	Time (secs)	0.150	0.203	0.437
CS_2	# Conflicts	1	26	221
	Time (secs)	0.406	5.751	45.682
CS_3	# Conflicts	26	43	535
	Time (secs)	10.961	20.901	378.178

Table 13: Performance of Diagnosis Algorithm.

We have conducted an empirical study (n=48 participants) related to the effectiveness of automated debugging in knowledge-based recommender application development (Felfernig 2006). The participants of the study had experiences related to the development and maintenance of knowledge-based recommender applications. Specifically, all participants had profound knowledge in constraint satisfaction problem solving and recommender technologies. The task of the participants was to identify faulty constraints in given constraint sets and to propose the corresponding repair actions. The major result of this study was that we can achieve significant time savings for error identification and repair tasks when applying automated debugging - for details see (Felfernig, 2006). Note that the study presented in (Felfernig 2006) did not focus on the debugging and repair of utility constraint sets, but investigated user performance when identifying and repairing faulty filter constraints in recommender knowledge bases. However, experiences from the application of our approach to real-world utility constraint sets clearly show similar positive effects in terms of reduced erroneous repair actions and significant time savings.

Related Work

There are three basic approaches to recommender application development. *Collaborative Filtering* (Herlocker et al. 2004) is based on the assumption of the correlation of preferences, i.e., similar products are recommended to customers with similar interests. *Content-based filtering* (Pazzani 1999) focuses on the analysis of a given set of products already purchased by the customer. By exploiting historical purchasing data, products are recommended which resemble those already purchased in the past. One of the major advantages of both approaches lies in the simple set-up procedure which does not require complex knowledge acquisition and maintenance tasks. In contrast, *knowledge-based* approaches rely on deep knowledge about the offered product and service assortments as well as deep knowledge about the company's marketing and sales strategy. In this context, the relationship between customer requirements and offered products has to be explicitly modelled in an underlying recommender knowledge base (Burke 2000, Felfernig and Kiener 2005). Deep knowledge representations are the major precondition for supporting explanations for recommendations, determining repair actions for inconsistent requirements, and applying Model-Based Diagnosis (MBD) techniques (Reiter 1987, de Kleer et al.1992) to the identification of faulty constraints in knowledge bases. The complexity of configuration knowledge bases motivated the application of MBD in knowledge-based systems development (Felfernig et al. 2004). Similar motivations led to the application of MBD in technical domains such as the development of hardware designs (Friedrich et al. 1999) or software development (Mateis et al. 2000). Compared to the work of (Felfernig et al. 2004), our work focuses on the maintenance of utility constraints – the major difference in this context lies in our diagnosis approach, where examples are not tested

sequentially w.r.t. their consistency with the knowledge base, but are tested within the scope of one theorem prover call. This approach is necessary since we are interested in repair actions (concrete instantiations) for utility constraints (scoring rules) consistent with all(!) given examples, whereas (Felfernig et al. 2004) did not calculate concrete repair actions for faulty constraints. (Biso et al. 2000) handle the problem of modelling real-life situations using soft constraints. In this context, solution preferences are treated as examples which are exploited for the learning of constraint preferences. Compared to our work, (Biso et al. 2000) focus on the learning of constraint preferences which allow the determination of appropriate results but do not explicitly take into account minimality aspects which allow us to determine minimal changes to existing utility constraint structures.

Conclusions

In this paper we have presented innovative knowledge acquisition techniques which effectively support knowledge engineers in the development and maintenance of constraint sets used for the calculation of product rankings. In order to be able to effectively adapt and to continuously improve existing utility constraint sets, we have integrated Model-based Diagnosis (MBD) concepts into recommender development environments where the generation of recommendations is based on explicit knowledge representations. The automated debugging of utility constraint sets is novel and is a general contribution to the effective development of applications based on explicit representations of recommendation knowledge.

References

Back T. and Schwefel H.P. An overview of evolutionary algorithm for parameter optimization. Evol. Comput. Volume 1, pp. 1-23, 1993.

Biso A., Rossi F., and Sperduti A. Experimental Results on Learning Soft Constraints, KR'02, pp. 435-444, 2000.

Burke R. Knowledge-based Recommender Systems. *Encyclopedia of Lib. and Information Syst*., 69(32), 2000.

Coulter K. and Coulter R. "Size Does Matter: The Effect of Magnitude Representation Congruency on Price Perceptions and Purchase Likelihood. *Journal of Consumer Psychology* 15,1, 64-76, 2005.

de Kleer J., Mackworth A. and Reiter R. Characterizing diagnoses and systems, *Artificial Intelligence* 56 (2-3) pp. 197-222, 1992.

Felfernig A., Friedrich G., Jannach D., Stumptner M. Consistency-based Diagnosis of Configuration Knowledge Bases. *Artificial Intelligence*, 2(152):213–234, 2004.

Felfernig A. and Kiener A. Knowledge-based Interactive Selling of Financial Services using FSAdvisor. IAAI'05, Pittsburgh, Pennsylvania, pp. 1475–1482, 2005.

Felfernig A. and Shchekotykhin K. Debugging User Interface Descriptions of Knowledge-based Recommender Applications. ACM IUI'06, pp. 234–241, 2006.

Felfernig A., Friedrich G., Teppan E. Automated Debugging of Utility Constraints in Recommender Knowledge Bases, University Klagenfurt, Technical Report, 01/2007.

Felfernig A. Reducing Development and Maintenance Efforts for Web-based Recommender Applications, *Intl. Journal of Web Engineering and Technology*, Mendes E. (Ed.), Special Issue on Empirical Web Engineering, 3(3):329-351, 2007.

Friedrich G., Stumptner M., and Wotawa F. Model-based diagnosis of hardware designs. *Artificial Intelligence* 111, 2, 3–39, 1999.

Gershberg F. and Shimamura, A. Serial position effects in implicit and explicit tests of memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20, 1370-1378, 1994.

Herlocker J., Konstan J., Terveen, L., and Riedl J. Evaluating Collaborative Filtering Recommender Systems. *ACM Transact. on Information Systems* 22, 1, 5–53, 2004.

Junker U. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. 19th National Conference on AI (AAAI'04), pp. 167–172, 2004.

Mateis C., Stumptner, M., and Wotawa, F. Modeling Java programs for diagnosis. 14th European Conference on Artificial Intelligence, 171–175, 2000.

Pazzani M. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review* 13, 5-6, 393–408, 1999.

Reiter R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence*, 23, 1, 57–95, 1987.

Winterfeldt D., Edwards W. Decision Analysis and Behavioral Research, Cambridge University Press, Cambridge, England, 1986.

Sensor placement for maximum fault isolability

Erik Frisk and Mattias Krysander
Dept. of Electrical Engineering, Linköping University
SE-581 83 Linköping, Sweden
{frisk,matkr}@isy.liu.se

Abstract

An algorithm is developed for computing which sensors to add to obtain maximum fault detectability and fault isolability. The method is based on only the structural information in a model which means that large and non-linear differential-algebraic models can be handled in an efficient manner. The approach is exemplified on a model of an industrial valve where the benefits and properties of the method is clearly shown.

1 Introduction

Fault diagnosis and process supervision is an increasingly important topic in many industrial applications and there are many publications in this area, see for example [Blanke *et al.*, 2003] and the references therein. To be able to perform model based supervision, some redundancy is needed and this redundancy is typically provided by sensors mounted on the process. Scientific attention has mainly been devoted to the design of a diagnosis system, given a model of a process equipped with a set of sensors. Not as much attention has yet been devoted to decide which sensors to include in the process.

There are many types of performance measures in diagnosis, for example detection performance, false alarm probabilities, time to detection etc. In this paper, sensors are placed such that maximum detectability and isolability is possible, i.e. faults in different components should, as far as possible, be able to be isolated from each other. Since sensor placement is often done early in the design phase, possibly before a reliable process model can be developed, the method developed in this paper is based on a structural process model. This is a coarse model description that can be obtained early and without major engineering efforts. Also, this means that large and non-linear differential-algebraic models can be handled in an efficient manner. The drawback with structural methods is that only best case results are obtained, see [Krysander, 2006] for a more on depth discussion on this.

2 Problem Formulation

Before the main objective of the paper is formally presented, a small example is discussed that illustrates the fundamental problems in sensor placement for fault diagnosis. The example is modeled by a fifth order linear system of ordinary differential equations. This example will be used throughout the paper, although the results will be equally applicable to large scale, non-linear, differential-algebraic models. The model consists of the following equations

$$\begin{aligned} e_1: &\quad \dot{x}_1 = -x_1 + x_2 + x_5 \\ e_2: &\quad \dot{x}_2 = -2x_2 + x_3 + x_4 \\ e_3: &\quad \dot{x}_3 = -3x_3 + x_5 + f_1 + f_2 \\ e_4: &\quad \dot{x}_4 = -4x_4 + x_5 + f_3 \\ e_5: &\quad \dot{x}_5 = -5x_5 + u + f_4 \end{aligned}$$

where x_i are the state variables, u a known control signal, and f_i the faults we want to detect and isolate. Since there are no specified sensors there is no redundancy and the faults are not detectable.

In this example, faults are modeled by fault signals that are included in the model equations and $f_i \neq 0$ indicates a fault. A more general way to include faults is to assign assumptions, or support, to the equations. This type of fault modeling can also easily be used with the approach that will be presented later but for sake of simplicity, fault signal modeling will be used in the paper. Also, from now on only single faults will be considered and f_i will then be used to denote both the fault signal and the fault mode.

Now, define *minimal sensor set* which is a minimal set of sensors to add to achieve maximum fault isolability.

Definition 1 (Minimal sensor set) *Let $\mathcal{S}$ be the set of possible sensor locations, i.e. the set of measurable variables, and let S be a multiset defined on $\mathcal{S}$. Then S is a minimal sensor set if adding the sensors in S give maximum fault isolability and all proper subsets of S do not.*

Note that S is a multiset, which is similar to a set but allows multiple instances of a member. Generalizations of the standard set operations like union and intersection are straightforward. Multisets are used instead of regular sets since it may be necessary to add more than one sensor measuring the same variable.

Returning to the example, a first question is then what are the minimal sensor sets achieving detectability of all faults? Here it is assumed that sensors measure a state-variable or a function thereof. It can be shown, using conditions for fault detectability in linear systems, that $\{x_1\}$, $\{x_2\}$, $\{x_3, x_4\}$ are all minimal sensor sets achieving detectability of all faults.

A second step is to not only require detectability, but also isolability properties. Here isolability refers to isolability as it is commonly used in FDI and the consistency based diagnosis AI community, see e.g. [Cordier *et al.*, 2004]. For details on how isolability is defined in this paper, see Sections 3 and 4. It can be shown that there are 5 minimal sensor sets that achieve maximal fault isolation: $\{x_1, x_3\}$, $\{x_1, x_4\}$, $\{x_2, x_3\}$, $\{x_2, x_4\}$, and $\{x_3, x_4\}$. Thus, adding sensors measuring the variables in any of these sets, or a superset of the variables, achieves maximum fault isolability.

Now, it is of course the case that the new sensors may also become faulty. If we want also faults in the new sensors to be isolable from the other faults we may have to add additional sensors. In this case, if maximum fault isolability is desired also for faults in the new sensors, there are 9 minimal sensor sets where one sensor set is two sensors measuring x_1 and one for x_3, i.e. the multiset $S = \{x_1, x_1, x_3\}$.

The problem formulation can now be stated as:

> Given a model and possible sensor locations, find all minimal sensor sets with the maximum possible fault isolability.

The methods developed in sections that now follow aim at addressing this problem for general, non-linear and differential-algebraic models. Doing this analytically is difficult since then inference concerning solutions to the model equations is needed. Instead, a method based on utilizing only the structure of the model is employed. This gives generic results that hold in a best-case situation. An advantage is that large models can be handled in an efficient manner. See Section 3 for some further results on the relation between structural and analytical properties of a model. See also [Krysander, 2006] for an in depth discussion on this topic.

3 Theoretical Background

The sensor placement problem will here be solved using a structural representation of the model. The structural representation of a set of equations M with unknown variables X is a bipartite graph with variables and equations as node sets. The known variables are, in this paper, omitted in the structure because they will not be needed for the analysis. There is an edge in the graph between a node representing an equation $e \in M$ and node representing an unknown variable $x \in X$ if the variable x is contained in e. A bipartite graph can be described by a biadjacency matrix where the rows and columns correspond to the node sets and the position (i, j) is one if there is an edge between node i and j, and a zero otherwise.

Structural methods can be applied to dynamical systems and the structure of the dynamic example formulated in Section 2 is shown in Figure 1 as a biadjacency matrix of the bipartite graph. The position (e_i, x_j) is one if x_j or any time-derivative appear in equation e_i. This structural representation of dynamical systems has been used in for example [Frisk *et al.*, 2003] and [Ploix *et al.*, 2005]. There exist other structural representations of dynamical systems, but the one used here is a compact representation suitable for the sensor placement problem [Krysander, 2006].

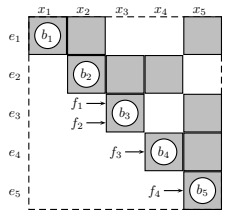


Figure 1: Structure of the linear example in Section 2. Gray areas indicate non-zero elements.

3.1 Dulmage-Mendelsohn decomposition

We will frequently use the Dulmage-Mendelsohn decomposition [Dulmage and Mendelsohn, 1958] which is illustrated in Figure 2. The decomposition defines a partition $(M_0, M_1, \ldots, M_n, M_\infty)$ of the set of equations M, a similar partition of the set of unknowns X, and a partial order on the sets M_i. If the rows and columns are rearranged according to this order, the biadjacency matrix has the form shown in Figure 2. There are zero entries in the white parts of the matrix and there might be ones in the gray-shaded parts. Three main parts of M can be identified in the partition, M_0 is called the structurally underdetermined part, $\cup_{i=1}^{n} M_i$ is the structurally just-determined part, and M_∞ is the structurally overdetermined part. In the figure, each pair (M_i, X_i) is related to a block which is denoted by b_i.

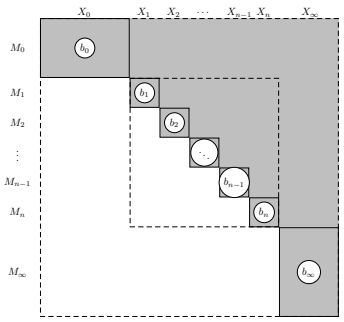


Figure 2: Dulmage-Mendelsohn decomposition

Diagnosis related treatments of Dulmage-Mendelsohn decomposition can be found in e.g. [Blanke *et al.*, 2003; Krysander, 2006].

3.2 Structural formulation of fault diagnosis

In this section, we will give structural characterizations of fault diagnosis properties. By doing this, the sensor placement problem can be formulated as a graph theoretical problem.

Let M denote a set of equations and F a set of single faults. Without loss of generality, it is possible to assume that a single fault can only violate one equation. If a fault signal f appears in more than one equation, we simply replace f in the equations with a new variable x_f and add equation $f = x_f$ which then will be the only equation violated by this fault. An example of this procedure is also given in the example in Section 5. Let $e_f \in M$ be the equation that might be violated

by a fault $f \in F$. For the example introduced in Section 2, $e_{f_1} = e_{f_2} = e_3$, $e_{f_3} = e_4$, and $e_{f_4} = e_5$.

An equation is, in the generic case, monitorable if it is contained in the structurally overdetermined part of M [Blanke *et al.*, 2003]. If the structurally overdetermined part of a set of equations M is denoted by M^+, then the structural characterization of detectability can be defined as follows.

Definition 2 *A fault f is structurally detectable in a model M if $e_f \in M^+$.*

Returning to the example and illustrating the correspondence between detectable faults and structurally detectable faults, assume that a sensor y measuring x_4 has been added to the process and included in the model by $e_6 : y = x_4$. The faults f_3 and f_4 are the detectable faults and a residual capable of detecting them is

$$r = 20y + 9\dot{y} + \ddot{y} - u = 5f_3 + \dot{f}_3 + f_4$$

which in fact is the only residual generator for this model modulo post filtering. Thus, faults f_1 and f_2 are not detectable. The structurally overdetermined part M^+ of the model $M = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ is equal to $\{e_4, e_5, e_6\}$. The equations $e_{f_3} = e_4$ and $e_{f_4} = e_5$ corresponding to the detectable faults f_3 and f_4 belong to M^+, but not the equations corresponding to the other faults. This implies according to Definition 2 that the detectable faults, f_3 and f_4, are the structurally detectable faults in M which is in accordance with the analytical result above.

Detection is a special case of isolation, i.e. a fault is detectable if the fault is isolable from the no-fault mode. By noting this similarity, it holds that a fault f_i, isolable from f_j, can violate a monitorable equation in the model describing the behavior of the process having a fault f_j. The equations valid with a fault f_j is $M \setminus \{e_{f_j}\}$ and the monitorable part of these equations is, in the generic case, equal to $(M \setminus \{e_{f_j}\})^+$. This motivates the following structural characterization of isolability.

Definition 3 *A fault f_i is structurally isolable from f_j in a model M if $e_{f_i} \in (M \setminus \{e_{f_j}\})^+$.*

The structural detectability and isolability definitions will next be used in a structural approach for solving the sensor placement problem.

4 A Structural Approach

Theoretical results and an algorithm to solve the problem posed in Section 2 is here formulated using the theory in Section 3. Due to space limitations, all proofs are omitted but can be found in [Krysander and Frisk, 2007].

A general assumption of the approach is that the model does not contain any underdetermined part. This is not a restrictive assumption since any complete physical model will, given an initial condition, have a unique solution and thereby no underdetermined part. Without loss of generality, it is also assumed that no fault affects more than one equation and that possible sensors measure a function of one unknown variable. In case there are possible sensors that measure some function h of more than one unknown variable, include a new equation $x_{new} = h(x)$ in the model.

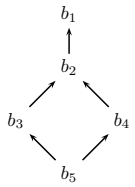


Figure 3: Hasse diagram of the partial order over the set of strongly connected components B.

4.1 Sensor placement for detectability

A basic building block in the final algorithm will be to find minimal sensor sets that achieve structural detectability of faults in an exactly determined set of equations. This section will be devoted to solving this sub-problem by first outlining the solution for the example system from Section 2 and then formally stating the solution. Although the example is given by analytical equations, all results in this section are based on the structural model only.

The example model is, without any additional sensors, an exactly determined set of equations with 5 equations and 5 unknown variables x_i, i.e. all faults are undetectable. Consider first the fault f_3. To make this fault detectable, according to Definition 2, an additional sensor is needed such that equation $e_{f_3} = e_4$ becomes a member of the overdetermined part of the model.

It is straightforward to verify that f_3 becomes detectable if and only if any of the variables $\{x_1, x_2, x_4\}$ are measured. For example, measuring x_4 makes the new measurement equation together with equations e_4 and e_5 an overdetermined set of equations. For this set of equations, a residual generator which is sensitive to fault f_3 can easily be derived. A similar line of reasoning can be made when measuring x_1 or x_2.

Then, why are x_1, x_2, and x_4 exactly those measurements that give detectability of f_3? The explanation can be seen in Figure 1 where it can be noted that block b_1 is connected with b_2 via a non-zero element in position $(1, 2)$ and in a similar fashion is b_2 connected to b_4. Thus, there is a connection between variables x_1, x_2, and x_4, which is precisely the variable in block b_4 including fault f_3. Measuring x_3, i.e. the variable in b_3, do not give detectability of f_3 since there is no connection between b_3 and block b_4.

The above reasoning indicates that some order between the strongly connected components might be useful. Let the exactly determined part of Figure 2 be the adjacency matrix of a directed graph on the set of strongly connected components b_i. A non-zero element in block (i, j) indicates a directed edge from b_i to b_j. A partial order on the blocks can then be defined as $b_i \leq b_j$ if and only if there is a path from b_i to b_j. Figure 3 shows the Hasse diagram of the ordering of the strongly connected components for the example.

With this ordering one can state exactly which parts of an exactly determined model that becomes overdetermined when adding a sensor. The following lemma formalizes the discussion above. This also gives, according to Definition 2, which faults that become detectable as a result of adding a sensor.

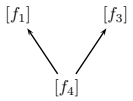


Figure 4: Hasse diagram of the partial order for the linear example over the set of fault classes. In Figure 1 it can be seen that $[f_1] = [f_2]$. Classes $[f_1]$ and $[f_3]$ are the maximal elements of the partial order.

Lemma 1 *Let M be an exactly determined set of equations, b_i a strongly connected component in M, and $e \notin M$ an equation corresponding to measuring any variable in b_i. Then*

$$(M \cup \{e\})^+ = \{e\} \cup (\cup_{j:b_j \leq b_i} M_j)$$

Achieving detectability of one fault affecting a strongly connected component immediately implies detectability of all faults affecting the same component. Therefore it makes sense to define an equivalence relation on the set of faults where all faults influencing the same strongly connected component are equivalent. A set of equivalent faults is denoted as $[f_i]$ where f_i is one element in the equivalence class. Now, based on Lemma 1, it is clear that measuring a variable in a block ordered higher than the block the fault enters achieves detectability. Let $P \subseteq X$ be a set of possible sensor locations and introduce the set

$$D([f_i]) = \{x | b_j \in B \wedge b_i \leq b_j \wedge x \in X_j \cap P\}$$

where X_j is the set of variables corresponding to block b_j, B the set of strongly connected components, and b_i the block that is influenced by the faults in $[f_i]$. The set $D([f_i])$ is thus the set of variables such that measuring *any* variable in the set achieves detectability of all faults in the equivalence class $[f_i]$.

Returning to the example and utilizing the result above, one can see that detectability of f_4 comes automatically when adding sensors to achieve detectability of either the faults in $[f_1]$ or $[f_3]$. This is because b_5 is less or equal than both b_3 and b_4 and according to Lemma 1, block b_5 is automatically included in any overdetermined set of equations when $[f_1]$ or $[f_3]$ are made detectable. This means that it is only necessary to ensure detectability for a subset of the fault classes to ensure detectability of all faults. To illustrate exactly which classes, introduce an order on the equivalence classes of F, defined as $[f_i] \leq [f_j]$ if $b_i \leq b_j$ where b_k is the block where the faults in $[f_k]$ enters the model. Figure 4 shows the Hasse diagram of the partial order for the example model. Here one can see that in the example it is necessary and sufficient to ensure detectability of the maximal elements of the partial order. In the example the set of possible sensor locations is X, but with a P that is a proper subset of X one might have the case where a maximal fault class is not detectable regardless of which sensors in P that is added. In such a case, one need to consider the maximal elements among the detectable fault classes.

The following theorem proves the general result and summarizes the discussion of this section.

Theorem 1 *Let M be an exactly determined set of equations, F the corresponding set of faults, $P \subseteq X$ the set of possible sensor locations, and M_S the equations corresponding to adding a set of sensors S. Then maximal detectability of faults F in $M \cup M_S$ are obtained if and only if S has a non-empty intersection with $D([f])$ for all $[f] \in F_m$ where F_m is the set of maximal fault classes among the fault classes with $D([f]) \neq \emptyset$.*

The above result can be summarized in an algorithm that given a model M, faults F, and a set of possible sensor locations P, computes the family of detectability sets $\mathcal{D}$.

```
1 function D = Detectability(M,F,P)
2   Compute block and fault class orders using M;
3   F_m = set of maximal fault classes [f] s.t. D([f]) ≠ ∅;
4   D={D([f])|[f] ∈ F_m};
```

Our objective was not to compute the set of detectability sets $\mathcal{D}$, but rather minimal sensor sets. For this, note that a hitting set for a family of sets is a set that has non-empty intersection with each set in the family. Thus, a minimal hitting set algorithm [Reiter, 1987; de Kleer, 1987] applied to the family of sets $\mathcal{D}$ can be used to efficiently find all minimal sensor sets.

For the example model, as previously noted, the maximal fault classes are $[f_1]$ and $[f_3]$ and the corresponding detectability sets are $D([f_1]) = \{x_1, x_2, x_3\}$ and $D([f_3]) = \{x_1, x_2, x_4\}$. Theorem 1 gives that the minimal sensor sets that achieve detectability of all faults are $\{x_1\}$, $\{x_2\}$, and $\{x_3, x_4\}$, which are the same sensor sets as was determined in Section 2.

4.2 Sensor placement for isolability of detectable faults

This section describes the basic ideas of how to find the minimal sensor sets such that maximum single fault isolability is obtained under the assumption that all faults are structurally detectable. In the next section this assumption will be removed.

The problem of achieving maximum isolability of the set of single faults F can be divided into $|F|$ sub-problems, one for each fault, as follows. For each fault $f_j \in F$, find all measurements that make the maximum possible number of faults $f_i \in F \setminus \{f_j\}$ isolable from f_j. The solution to the isolability problem will then be obtained by combining the results from all sub-problems.

Each sub-problem can be formulated as a detectability problem, as will be shown next. Assume that M is a model, including sensors such that all faults are detectable, and M_S represents a set of equations describing an additional sensor set S. Given the sensor set S, a fault f_i is isolable from f_j in the model $M \cup M_S$ if $e_{f_i} \in ((M \cup M_S) \setminus \{e_{f_j}\})^+$ according to Definition 3. By introducing $M' = M \setminus \{e_{f_j}\}$, this can be written as

$$e_{f_i} \in (M' \cup M_S)^+ \quad (1)$$

which according to Definition 2 means that f_i is structurally detectable in $M' \cup M_S$. Hence the maximum possible number of faults $f_i \in F \setminus \{f_j\}$ are isolable from f_j in $M \cup M_S$ if the maximum possible number of faults $f_i \in F \setminus \{f_j\}$ are structurally detectable in the model $(M \cup M_S) \setminus \{e_{f_j}\}$. This shows that each sub-problem can be formulated as a detectability problem.

Next, we use the example formulated in Section 2 to outline the solution of one sub-problem before formally stating

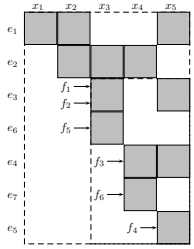


Figure 5: Block structure of the example in Section 2 extended with measurements of x_3 and x_4.

the solution. Assume that we have added sensors measuring $\{x_3, x_4\}$ such that all faults are detectable. Furthermore, assume that these sensors can be faulty and denote these faults f_5 and f_6 respectively. A row permuted structure of the obtained model $M = \{e_1, e_2, \ldots, e_7\}$ is shown in Figure 5.

Consider the sub-problem associated with fault f_1. The set M' in (1) is equal to $M \setminus \{e_{f_1}\} = M \setminus \{e_3\}$. The sub-problem is, given the model $M \setminus \{e_3\}$, to find the minimal additional sensor sets S such that as many of the faults $f_2, f_3, \ldots, f_6$ as possible become detectable in $M' \cup M_S$.

The fault f_2 is not included in M' and cannot be structurally detectable in $M' \cup M_S$ for any sensor set S. This implies that f_2 is not isolable from f_1 with any sensor addition and this also follows from the fact that these to faults violate the same equation. The faults f_3, f_4, and f_6 in the structurally overdetermined part $(M')^+ = \{e_4, e_5, e_7\}$ are according to Definition 2 structurally detectable in M' and require no additional measurements. Fault f_5 in the just-determined part $\{e_1, e_2, e_6\}$ is not detectable, but f_5 can become detectable in $M' \cup M_S$ if S is appropriate selected.

Sufficient and necessary requirements on S can be computed by the function `Detectability` described in Section 4.1. By applying this function to the structurally just-determined part of M', i.e. the sub-graph of M' defined by the node sets $\{e_1, e_2, e_6\}$ and $\{x_1, x_2, x_3\}$, we get that $D([f_5]) = \{x_1, x_2, x_3\}$. Hence, one of the variables in the detectability set $\{x_1, x_2, x_3\}$ must be measured to make the faults $F \setminus \{f_1, f_2\}$ detectable in $M' \cup M_S$ and this implies that all faults in $F \setminus \{f_1, f_2\}$ are isolable from f_1 in $M \cup M_S$. The solution to the sub-problem related to fault f_1 will be the computed detectability set. The next theorem formalizes the solution of a sub-problem like the one discussed above.

Theorem 2 *Let M be a set of equations with no structurally underdetermined part, F a set of structurally detectable faults in M, $P \subseteq X$ the set of possible sensor locations, and M_S the equations added by adding the sensor set S. Let M^0 be the just-determined part of $M \setminus \{e_{f_j}\}$, F^0 the faults contained in M^0, and $\mathcal{D} =$ `Detectability`(M^0, F^0, P). Then the maximum possible number of faults $f_i \in F \setminus \{f_j\}$ are structurally isolable from f_j in $M \cup M_S$ if and only if S have a non-empty intersection with all sets in $\mathcal{D}$.*

The result of the theorem can be summarized in a function that given a model M, a set of detectable faults F in M, the possible sensor locations P, and a fault $f \in F$, computes the family of detectability sets $\mathcal{D}$ that solves the isolability sub-problem for f.

```
1 function D = IsolabilitySubProblem(M,F,P,f)
2   M^0 = just−determined part of M \ {e_f};
3   F^0 = the set of faults F included in M^0;
4   D = Detectability(M^0,F^0,P);
```

An additional sensor set that maximizes the set of fault pairs (f_i, f_j) such that f_i is structurally isolable from f_j must have a non-empty intersection with all detectability sets found in all sub-problems.

```
1 function D = Isolability(M,F,P)
2   D = ∅;
3   for f_i ∈ F
4     F' = F \ {f_i};
5     D = D ∪ IsolabilitySubProblem(M,F',P,f_i);
6   end
```

The minimal sensor sets that maximize the isolability can be found by applying a minimal hitting set algorithm to the sets in the output $\mathcal{D}$.

For the example shown in Figure 5, the families of detectability sets of the different sub-problems are $\{\{x_1, x_2, x_3\}\}$ for f_1, f_2, and f_5, $\{\{x_1, x_2, x_4\}\}$ for f_3 and f_6, and $\varnothing$ for f_4. We have found two distinct detectability sets and the minimal hitting sets are $\{x_1\}$, $\{x_2\}$, and $\{x_3, x_4\}$. These sets are the minimal additional measurements that achieve maximum single fault isolability.

4.3 Sensor placement for both detectability and isolability

We have shown how isolability can be achieved in a model where all faults are structurally detectable. Next, we will extend the presented solution to models where faults may not be structurally detectable in the original model.

The solution is first outlined for the example described in Section 2. The faults in this model are not detectable and we want to find all minimal sensor sets that maximize fault detectability and isolability. We have shown in Section 4.1 that the minimal sets of measurements to achieve full detectability are $\{x_1\}$, $\{x_2\}$, and $\{x_3, x_4\}$. If we add for example a sensor measuring x_1 described by an equation e_s, we get a new model $M \cup \{e_s\}$ where all faults are detectable. Since all faults are detectable, the previously described method to achieve maximum isolability can be applied to the model $M \cup \{e_s\}$. The minimal sensor sets that solve this problem are $\{x_3\}$ and $\{x_4\}$. By combining this result with the fact that a sensor measuring x_1 has been added to obtain detectability, it follows that $\{x_1, x_3\}$ and $\{x_1, x_4\}$ are two possible sensor sets that achieve maximum detectability and isolability. To compute all minimal sensor sets that achieve maximum isolability, we also have to investigate the solutions when we choose to measure $\{x_2\}$ or $\{x_3, x_4\}$ to obtain full detectability. By solving one isolability problem for each of the minimal sensor sets that achieves full detectability, we get that the minimal sensor sets are $\{x_1, x_3\}$, $\{x_1, x_4\}$, $\{x_2, x_3\}$, $\{x_2, x_4\}$, and $\{x_3, x_4\}$ which are the same sets as in Section 2.

The following description summaries the suggested algorithm that given a model M with no structurally underdetermined part, the faults F, and the possible sensor locations P, computes the family $\mathcal{S}$ of all minimal sensor sets that

achieve maximum isolability. In the algorithm the join operation of two multisets A and B will be used. The join operation is denoted by $A \uplus B$ and is defined as the multiset containing all elements in $A \cup B$ with a multiplicity equal to the sum of the multiplicities in A and B. For example $\{x_1, x_2\} \uplus \{x_1\} = \{x_1, x_1, x_2\}$.

```
1  function S = SensorPlacement(M,F,P)
2    S = ∅;
3    M^0 = just−determined part of M;
4    F^0 = the set of faults F included in M^0;
5    D = Detectability(M^0,F^0,P);
6    S_d = MinimalHittingSets(D);
7    for S_i ∈ S_d
8      Create the extended model M_e = M ∪ M_Si;
9      F_e = the faults included in M_e;
10     D = Isolability(M_e,F_e,P);
11     S_i = MinimalHittingSets(D);
12     S = S ∪ {S_i ⊎ S'|S' ∈ S_i};
13   end
14   Delete non−minimal sensor sets in S;
```

4.4 Adding sensors with faults

Sensors might have corresponding sensor faults. When adding a sensor, it is possible that a new fault is introduced into the model and in this section it is shown how these additional sensor faults can be handled.

Consider again the example introduced in Section 2 and assume now that we want to find all minimal sensor sets that maximize the fault isolability when all sensors introduce new possible faults. To do this, we will follow the algorithm `SensorPlacement` and describe how some of the lines should be modified to cope with additional sensor faults.

The additional sensors that have a corresponding sensor fault have to be specified in the algorithm. This is done by introducing an additional input set $P_f \subseteq P$ where sensors measuring variables in P_f may become faulty and the other sensors may not. For the example P_f is equal to P.

The purpose of line 5 and 6 is to compute all sensor sets that achieves full detectability. In Section 4.1, it was shown that $\{x_1\}$, $\{x_2\}$, or $\{x_3, x_4\}$ are the minimal sensor sets that make the original faults $f_1, \ldots, f_4$ detectable and the next theorem shows that all additional sensors faults will automatically become detectable.

Theorem 3 *Let M be a model with no underdetermined part and let x be a measured variable with a sensor described by an equation $e \notin M$. Then, a sensor fault violating e will be structurally detectable in $M \cup \{e\}$.*

The result of this theorem is for the example that $\{x_1\}$, $\{x_2\}$, and $\{x_3, x_4\}$ are the minimal sensor sets that make all faults, including the new faults introduced by the added sensors, detectable. Hence the inputs to the function `Detectability` described in Section 4.1 do not need to be changed at all.

On line 7, a minimal sensor set S_i that achieves full detectability is selected and on line 8 the equations M_{S_i} are added to the original model to form the extended model M_e. If the sensors may become faulty, i.e. if $s \in S_i$ belongs to P_f, then these faults must be added to the model as done in Figure 5. These faults and the original faults in F are then stored on line 9 in F_e.

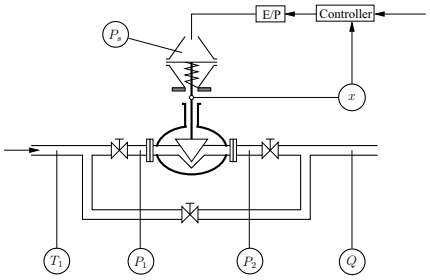


Figure 6: DAMADICS valve

The purpose of lines 10-11 is to, given the extended model M_e, find the family $\mathcal{S}_i$ of all minimal additional sensor sets S' achieving maximum isolability among both the faults in F_e and the sensor faults associated with the additional sensors S'. The next result states that if S' achieves maximum isolability among the faults F_e, then S' also achieves the maximum isolability among all faults including the faults introduced by the sensors in S'.

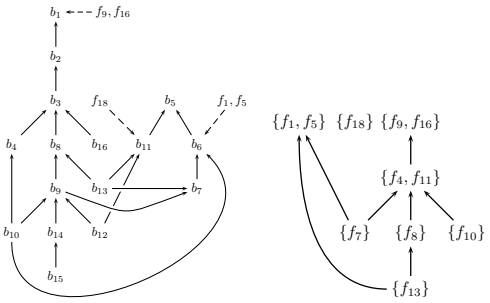
Theorem 4 *Let M be a model with no underdetermined part and F a set of structurally detectable faults in M. Furthermore, let M_S be an equation set describing additional sensors and F_S the associated set of sensor faults. Then for any sensor fault $f \in F_S$ and for any fault $f' \in (F \cup F_S) \setminus \{f\}$, it holds that f is isolable from f' and f' is isolable from f in $M \cup M_S$.*

The theorem shows that once sensors and sensor faults have been added to the original model on line 8, the minimal additional sensor sets to achieve maximum isolability can be computed exactly as before, i.e., the lines 10-14 need not be changed. In conclusion, the only difference in the function `SensorPlacement` when considering sensor faults is to add the additional input P_f that should be used in the creation of the extended model M_e on line 8.

A difference in the result from the case when not considering sensor faults is that the solution might include two sensors measuring the same variable. For the example, the minimal sensor sets when considering sensor faults are $\{x_1, x_1, x_3\}$, $\{x_1, x_1, x_4\}$, $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$, $\{x_1, x_3, x_4\}$, $\{x_2, x_2, x_3\}$, $\{x_2, x_2, x_4\}$, $\{x_2, x_3, x_4\}$, and $\{x_3, x_3, x_4, x_4\}$.

5 Example

The example used to illustrate the results is an industrial valve. A schematic figure of the valve is shown in Figure 6 and consists of three main components: the control valve, a by-pass valve, and a spring-and-diaphragm pneumatic servomotor to operate the valve plug. The figure also shows an internal control loop that is used to increase the accuracy of the valve plug positioning. Details of this model is not included in this presentation and interested readers are referred to e.g. [Syfert *et al.*, 2003] and the references therein. The structure of the model that is used here is derived in [Düştegör *et al.*, 2006].



(a) Order among strongly connected components. It is also noted, with dashed arrows, where some important faults appear in the model.

(b) Order among the equivalence classes on the set of faults.

Figure 7: Order among strongly connected components and faults for the Damadics valve model.

The original model included a specified set of sensors but since the objective here is to perform sensor placement analysis, almost all sensors has been removed. Three sensors has been kept, measurements of the two ambient pressures P_1, P_2, and the measurement of the valve position x that is used in the internal control loop. A fault, denoted f_{10} in the model, influences 2 equations and therefore a dummy variable x_{f10}, and an equation $x_{f10} = f_{10}$ is introduced to ensure that the assumption that each fault only influences 1 equation holds. In this example, all unknown variables *except* the dummy variable x_{f10} are assumed to be possible sensor locations. Of course no fault variables f_i can be measured. This leaves us with a model, which has no underdetermined part, consisting of 17 equations in 16 unknown variables and 12 different faults.

First, to determine which sensors that are necessary to obtain detectability of all faults, the partial orders on the strongly connected components and the fault equivalence classes are computed. Figure 7 shows the Hasse diagrams for both partial orders. In Figure 7-b it is clear that there are three maximal elements of the order, $\{f_1, f_5\}$, $\{f_{18}\}$, and $\{f_9, f_{16}\}$. Thus, obtaining detectability of these faults will automatically provide detectability of all other faults. It is noted in Figure 7-a which strongly connected components the maximal faults influence. Theorem 1 then gives that a sensor set achieving detectability has a non-empty intersection $D([f])$ for each maximal fault class. The unknown variables that appear in each relevant strongly connected component are $X_1 = \{P_z\}$, $X_5 = \{Q\}$, $X_6 = \{Q_v\}$, $X_{11} = \{Q_{v3}\}$ and then $D(\{f_1, f_5\}) = \{Q, Q_v\}$, $D(\{f_{18}\}) = \{Q, Q_{v3}\}$, and $D(\{f_9, f_{16}\}) = \{P_z\}$. By computing minimal hitting sets for these three sets one obtains two minimal sensor sets $\{P_z, Q\}$ and $\{P_z, Q_v, Q_{v3}\}$ and it can be verified using Definition 2 that all faults then become detectable.

Adding any of the above set of sensors only achieves detectability of the faults and does not give full isolability. Running the algorithm from Section 4.4, computing sensor sets that achieves maximum isolability also for faults in the new sensors, gives 8 minimal sensor sets. The minimal sensor sets has 7 or 8 sensors and one minimal set is to add sensors measuring the variables $\{P_s, P_z, P_z, Q, Q, Q_{v3}, x\}$. Note here that we need to add 2 sensors each for the variables P_z and Q. With these sensors, all faults are isolable from each other except for the pairs $\{f_4, f_{11}\}$, $\{f_1, f_5\}$, and $\{f_9, f_{16}\}$. This is because these faults cannot be isolated by adding more sensors measuring unknown variables since they appear in the same equation in the model. The only solution is to do further fault modeling [Frisk *et al.*, 2003] or, possibly rather unrealistic, include a sensor that measures the fault signal directly as in [Commault *et al.*, 2006].

6 Related Work

Sensor placement for diagnosis and fault detection is a well studied problem considering many different aspects. This discussion on relations to other works will focus on three papers that all have problem formulations with strong similarities to this paper.

In [Commault *et al.*, 2006] the sensor placement problem is addressed using input-output separators in a graph-based representation of the system model. A main difference to our paper is that Commault et.al. aims at adding sensors such that, in the linear case, it is possible to obtain a diagonal transfer matrix from faults to residual. This is often a rather unrealistic goal since this is only possible if there are more sensors than faults and for example if the added sensors may become faulty it is generically not possible to solve the posed problem. In addition it is in the paper assumed that fault signals can be measured which is an unrealistic assumption.

The basic problem formulation in [Raghuraj *et al.*, 1999] is almost identical to our paper but the model description is a little bit different. It is a graph-based description and they do not allow cycles in the graph and this results in loss of isolability performance in the solution. A drawback with their proposed solution is that their algorithm does not find all minimal sensor sets, the result does not even need to be minimal. However, it should be possible to use a minimal hitting set algorithm, instead of their greedy search, to obtain all minimal solutions to their posed problem. Another pair of differences are that they do not consider faults in the added sensors and also that faults entering in more than one equation is treated in a non-standard way. For example, in their approach it is not possible to add sensors such that the faults in the model

$$\dot{x} = Ax + \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} f$$

are isolable which is clearly possible.

A third related work is [Travé-Massuyès *et al.*, 2006] where the problem is approached by hypothesizing sensors and then computing the set of analytical redundancy relations (ARR), using all possible causalities, tracing the support of each ARR and then obtain isolability properties of the model. Travé-Massuyès et.al. assumes exoneration, i.e. that a fault always makes the corresponding residuals to exceed their thresholds, which is not assumed in our paper since this is a rather unrealistic assumption. Our approach computes which sensors to add to obtain a certain isolability performance while [Travé-Massuyès *et al.*, 2006] does it the other way around, adding all possible sensors and then removing

sensors until isolability performance decreases. One can expect severe complexity problems with such an approach since the number of ARR:s is exponential in the redundancy of the model [Krysander *et al.*, 2007] and by adding all possible sensors you obtain maximum redundancy. Another difference worth noting is that the performance measure in their paper is a scalar value, the diagnosability degree, equal to the quotient of the number of fault classes by the number of faults. However, different sensor setups may have different isolability properties and still have the same diagnosability degree. This is the reason why the complete isolability relation, rather than e.g. the diagnosability degree, is used as a performance measure in our paper. Similar to our paper, Travé-Massuyès et.al. also includes the case where the new sensors also may become faulty. However, typically this also means that you may have to add more than one sensor to a specific variable and this is not covered in [Travé-Massuyès *et al.*, 2006] indicating possibly incomplete results.

7 Conclusions

The sensor placement problem has been addressed in this paper. The objective is to add sensors such that maximum fault isolability can be achieved. It is often the case that some process variables cannot be measured and this information need to be considered in an analysis. In addition, new sensors may of course also become faulty and these faults must also be included in the analysis. Typically, this means that more than one sensor have to be added measuring a specific signal.

A key contribution is a new algorithm for sensor placement that cope with all aspects mentioned above. Given a model, the possible sensor locations and a specification of which sensors that may be faulty the algorithm computes all minimal sensor sets that make, as far as possible, faults isolable from each other. Typically there is a cost associated with each type of sensor, for example price, maintenance cost, reliability etc. This means that the sensor set with the least number of sensors may not be the best choice. Since the result of the algorithm contain *all* minimal sensor sets, it is straightforward to pose an optimality condition regarding cost to find the best choice of sensors to add.

The algorithm has been applied to a non-trivial industrial valve model with 17 equations and 15 possible sensor positions. The result was 8 minimal sensor sets that achieve maximum isolability also for faults in the added sensors. The minimal sensor sets have 7 or 8 sensors and several of them contain 2 sensors at the same position. A Matlab implementation of the algorithm is available at `http://www.fs.isy.liu.se/Software/SensPlaceTool/`.

References

[Blanke *et al.*, 2003] M. Blanke, M. Kinneart, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer-Verlag, 2003.

[Commault *et al.*, 2006] C. Commault, J. Dion, and S.Y. Agha. Structural analysis for the sensor location problem in fault detection and isolation. In *Proceedings of IFAC Safeprocess'06*, Beijing, China, 2006.

[Cordier *et al.*, 2004] M.O. Cordier, P. Dague, F. Levy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transaction on Systems, Man, and Cybernetics – Part B*, 34(5):2163–2177, 2004.

[de Kleer, 1987] J. de Kleer. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Düştegör *et al.*, 2006] Dilek Düştegör, Erik Frisk, Vincent Coquempot, Mattias Krysander, and Marcel Staroswiecki. Structural analysis of fault isolability in the DAMADICS benchmark. *Control Engineering Practice*, 14(6):597–608, 2006.

[Dulmage and Mendelsohn, 1958] A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

[Frisk *et al.*, 2003] Erik Frisk, Dilek Düştegör, Mattias Krysander, and Vincent Cocquempot. Improving fault isolability properties by structural analysis of faulty behavior models: application to the DAMADICS benchmark problem. In *Proceedings of IFAC Safeprocess 03*, Washington, USA, 2003.

[Krysander and Frisk, 2007] M. Krysander and E. Frisk. Some theoretical results on sensor placement for diagnosis based on fault isolability specifications. Technical Report LiTH-R-2770, ISY, Linköping, Sweden, 2007. `http://www.fs.isy.liu.se/Publications/`.

[Krysander *et al.*, 2007] Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal over-constrained sub-systems for model-based diagnosis. *Accepted for publication in IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 2007.

[Krysander, 2006] Mattias Krysander. *Design and Analysis of Diagnosis Systems Using Structural Methods*. PhD thesis, Linköpings universitet, June 2006.

[Ploix *et al.*, 2005] S. Ploix, M. Desinde, and S. Touaf. Automatic design of detection tests in complex dynamic systems. In *Proceedings of 16th IFAC World Congress, Prague*, Prague, Czech Republic, 2005.

[Raghuraj *et al.*, 1999] R. Raghuraj, M. Bhushan, and R. Rengaswamy. Locationg sensors in complex chemical plants based on fault diagnostic observability criteria. *AIChE*, 45(2):310–322, 1999.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Syfert *et al.*, 2003] M. Syfert, M. Bartys, J. Quevedo, and R.J. Patton. Development and application of methods for actuator diagnosis in industrial control systems (damadics): A benchmark study. In *Proceedings of IFAC Safeprocess 03*, Washington, USA, 2003.

[Travé-Massuyès *et al.*, 2006] L. Travé-Massuyès, T. Escobet, and X. Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transaction on Systems, Man, and Cybernetics – Part A*, 36(6):1146–1160, 2006.

Modeling and Solving Diagnosis of Discrete-Event Systems via Satisfiability

Alban Grastien
NICTA &
Australian National University
Canberra, Australia

Anbulagan
NICTA &
Australian National University
Canberra, Australia

Jussi Rintanen
NICTA &
Australian National University
Canberra, Australia

Elena Kelareva
University of Melbourne
Melbourne, Australia

Abstract

The diagnosis of a discrete-event system is finding out whether the behavior of the system is normal or faulty, given observations of this behavior. We show how the diagnosis problems can be translated into the propositional satisfiability problem (SAT) and then solved by the state-of-the-art SAT algorithms. Our experiments demonstrate that the SAT algorithms are able to deal with the problems, which are hard for traditional diagnosis algorithms.

Introduction

The diagnosis of a discrete-event system is finding out whether the behavior of the system is normal or faulty, given observations of this behavior. This computation can be done by checking whether there exist normal/faulty behaviors on the model of the system that are consistent with the observations. The main difficulty of this task is that the behavior of the system is only partially observable.

Two types of approaches for solving this problem have been presented. In the first approach the system model is compiled off-line in a structure which can be used on-line for efficiently detecting whether an observation sequence corresponds to normal or faulty behaviors. The main example is the Sampath *diagnoser* (Sampath *et al.* 1995). However, the size of the diagnoser can be double exponential in the size of the system description in the worst case (Rintanen 2007), which can make the diagnoser practically impossible to generate. The second approach is based on the computation of all behaviors and on checking whether these behaviors are correct (Lamperti & Zanella 2003; Pencolé & Cordier 2005; Su & Wonham 2005; Jiroveanu & Boël 2005; Cordier & Grastien 2007). Computing all behaviors can be extremely expensive.

Basically, the diagnosis task involves finding paths in the model of the system. An efficient approach to solving similar path finding problems most notably in classical AI planning (Kautz & Selman 1996) and model-checking (Biere *et al.* 1999) is to reduce them to the satisfiability (SAT) problem in the classical propositional logic. Rintanen and Grastien (2007) have shown that a system can be proved non-diagnosable with this approach. Diagnosis of static systems has earlier been viewed as a logical satisfiability (consistency) problem and specifically as a SAT problem (Reiter 1987; Veneris 2003). Williams and Nayak (1996) proposed the use of propositional logic to determine the state of the system. Schumann *et al.* (2007) have investigated the use of logic to solve diagnosis problems. In this paper, the diagnosis task is translated to a SAT problem which then can be efficiently solved by the current state-of-the-art SAT solvers.

The structure of the paper is as follows. First we present a simple system that is for earlier approaches to diagnosis too difficult because of the very high number of trajectories and sets of possible current states. Then we present a formalization of the discrete-event system diagnosis problem and its translation into SAT. Extensions for more complex observations and for a more accurate diagnosis are given in the next two sections. In the experiments' section we present data about the computational properties of the state-of-the-art SAT solvers in solving the diagnosis problem and compare the proposed method with existing diagnosis approaches.

Example

We consider a system of 20 identical components in a 5×4 grid such that each component is connected to its 4 neighbors. Corner and border components are neighbors to corner and border elements in the opposite sides. For example, the component in position $(0, 2)$ is connected to components in positions $(0, 1)$, $(0, 3)$, $(1, 2)$ and $(4, 2)$. The behavior of each component is represented by the automaton in Figure 1. All the components are initially in the state O. When a failure occurs in a component, its state changes to F and the component sends the message *reboot!* to its neighbors which receive the message *reboot?*, leading to state W, FF or R depending on their current state.

The goal is to monitor this system. The events IReboot and IAmBack correspond to a component sending an alarm. Given these observations, the monitoring system must determine what happened in the system. Sampath *et al.* (1995) have proposed a method for solving this kind of problems. The method consists of compiling the system description to a finite state machine called a diagnoser which efficiently maps a sequence of observations to abstract representations of sets of possible current states. The main problem with this approach is that the size of the diagnoser can be exponential in the number of states, and for this reason it cannot be computed for many systems with more than a couple of hundred or thousands of states.

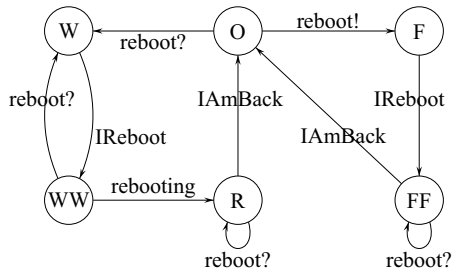


Figure 1: The behavior of one component

The other method involves computing all the behaviors of the system description that are consistent with the observations. However, for this example the number of such behaviors is astronomically large. We propose to solve the diagnosis problem by translating it into a SAT problem and then using state-of-the-art SAT solvers.

Definitions

System

We consider discrete-event systems, i.e. systems whose states can be represented by the assignment of a finite set of variables in finite domains. Here, we are restricted to two-valued (Boolean) state variables but in general variables may have any number of different values. Hence a state $s : A \rightarrow \{0, 1\}$ is a total function from the state variables to the constants 1 (*true*) and 0 (*false*). A *literal* is a state variable or its negation, and the set of all literals is $L = A \cup \{\neg a \mid a \in A\}$. The language $\mathcal{L}$ over A consists of all formulae that can be formed from A and the connectives $\vee$ and $\neg$. We use the standard definitions of further connectives $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$ and $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

Definition 1 (Model of the system)**.**
The model of the system *is a tuple* $SD = \langle A, \Sigma_u, \Sigma_o, \delta, s_0\rangle$ *where*

- A *is a finite set of* state variables*,*
- Σ_u *is a finite set of* unobservable events*,*
- Σ_o *is a finite set of* observable events*,*
- $\delta \subseteq \Sigma_o \cup \Sigma_u \rightarrow 2^{\mathcal{L}\times 2^L}$ *assigns each event a set of event instances* $\langle\phi, c\rangle$*, and*
- $s_0 : A \rightarrow \{0, 1\}$ *is the initial state.*

An *event instance* of the event e is a pair $\langle\phi, c\rangle \in \delta(e)$ which indicates that the event e can be associated with changes c in states that satisfy the condition ϕ. More formally, an event $e \in \Sigma_o \cup \Sigma_u$ is possible in any state s such that $s \models \phi$ for some $\langle\phi, c\rangle \in \delta(e)$. When e takes place in s, one of the pairs $\langle\phi, c\rangle \in \delta(e)$ satisfying $s \models \phi$ is chosen and the effect of the event is that the literals in c become *true*.

Let s be a state and $c \subset L$ a consistent set of literals (i.e. such that $l \in c \Rightarrow \neg l \notin c$). Then define the successor state $s' = \mathit{succ}(s, c)$ of s by

1. $s'(a) = 1$ for all $a \in A$ such that $a \in c$,
2. $s'(a) = 0$ for all $a \in A$ such that $\neg a \in c$, and
3. $s'(a) = s(a)$ otherwise.

The transition system is initially in the state s_0, and an event sequence $e_0, \ldots, e_{n-1}$ takes the system through a sequence $s_0, s_1, \ldots, s_n$ of states such that $\forall i \in \{0, \ldots, n-1\}$, $\exists\langle\phi, c\rangle \in \delta(e_i) : s_i \models \phi \wedge s_{i+1} = \mathit{succ}(s_i, c)$. Note that a state s_i and an event e_i do not necessarily determine the successor state s_{i+1} uniquely. The sequence of states and events is called a *trajectory* and models a behavior on the system.

This system description is very similar to the one used in planning. An important factor of efficient SAT-based planning (Kautz & Selman 1996) is the notion of parallel or partially ordered plans which allows several independent actions to be taken simultaneously. This approach has the advantage of making unnecessary to consider all $n!$ total orderings of n independent events, since their mutual ordering does not matter. The pairs $\langle\phi_1, c_1\rangle$ and $\langle\phi_2, c_2\rangle$ interfere if there is $a \in A$ that occurs positively/negatively in c_1 and negatively/positively in ϕ_2 or in c_2, or positively/negatively in c_2 and negatively/positively in ϕ_1. Events $e_1, \ldots, e_n$ can take place simultaneously with $o_1 \in \delta(e_1), \ldots, o_n \in \delta(e_n)$ if o and o' do not interfere for any $\{o, o'\} \subseteq \{o_1, \ldots, o_n\}$ such that $o \neq o'$.

We consider sequences $O_0, \ldots, O_{n-1}$ of (possibly empty) sets O_i of event occurrences, corresponding to the sets E_i of events, such that all members of O_i are mutually non-interfering. We define the successor state s' of s with respect to a set O of non-interfering event occurrences by

$$s = \mathit{succ}(s, \bigcup_{\langle\phi,c\rangle\in O} c).$$

Observations

The event sequence that occurs in the system is partially observable. Indeed, each observable event generates an observation emitted by the system. In general, it is considered that the observations received for the diagnosis are identical to the observations emitted by the system. However recent works (Lamperti & Zanella 2003; Grastien, Cordier, & Largouët 2005; Pencolé & Cordier 2005) consider uncertainties to do with the observations: partial order between the occurrence of observable events, uncertainty over which event has occurred, loss of observations, etc. To simplify, we consider in this section and the next one that the observations are a collection of timed observable events. This hypothesis is communly used in time-driven systems. More general cases are presented in a next section.

Definition 2 (Observations)**.**
The observations OBS *is a set of pairs* $\langle e, t\rangle$ *where* $e \in \Sigma_o$ *is an observable event and* $t \in \mathbf{N}^+$ *is a positive integer.*

The semantics is simple: if $\langle e, t\rangle \in OBS$, then the observable event e occurred at time step t. Conversely, if $\langle e, t\rangle \notin OBS$, the observable event e did not occur at time step t. A sequence of sets of events $E_0, \ldots, E_{n-1}$ is consistent with the observations OBS if:

$$\langle e, i\rangle \in OBS \Leftrightarrow (i < n \wedge e \in E_i) \quad \forall i \in \mathbf{N}^+, \forall e \in \Sigma_o.$$

Diagnosis

The *diagnosis label* of a trajectory can be *normal* or *faulty*. In this section, we consider that a trajectory is faulty if it contains a *faulty event*. The set of faulty events is denoted by $\Sigma_f \subseteq \Sigma_u$. More general cases are presented in section *Dealing with faulty behaviors*. Two trajectories are said to be *f-equivalent* if they share the same diagnosis label.

The behavior of the system is often ambigious: given a flow of observations of system behavior, the correctness of the behavior cannot always be proved, particularly if the system is not *diagnosable* (Sampath *et al.* 1995; Rintanen & Grastien 2007). Moreover, there is often a delay between a fault and the observations that enable this fault to be diagnosed. Because of this uncertainty, we consider that it is sufficient to find one normal behavior consistent with the observations to diagnose that the system behavior is correct.

Definition 3 (Diagnosis)**.**
Let SD be the model of the system, and let OBS be the observations on the system. The behavior of the system is normal *if there exists a sequence $\mathcal{E} = E_0, \ldots, E_{n-1}$ of events on SD consistent with OBS such that $\mathcal{E}$ is normal.*

As we show in section *Dealing with faulty behaviors*, the approach developed in this paper enables to answer *yes* or *no* to nearly any question about the behavior of the system and return a proof if the answer is *yes*.

Diagnosis by SAT

Diagnosing a system requires to find whether there exists a normal path on the model of the system that is consistent with the observations. This test can be formulated as a SAT problem, similarly to other path finding problems in AI planning (Kautz & Selman 1996).

We construct a formula such that satisfiable valuations of this formula correspond to the sequences $(s_0, \ldots, s_n)$ of states and the sequences $(E_0, \ldots, E_{n-1})$ of events consistent with the observations. The number n is the maximum value such that $\langle e, n-1\rangle \in OBS$.

Next, we define the formula Φ_{SD} that represents the system description $SD = \langle A, \Sigma_u, \Sigma_o, \delta, s_0\rangle$. The propositional variables, with superscript t corresponding to time step t, are the following:

- a^t for all $a \in A$ and $t \in \{0, \ldots, n\}$,
- e^t for all $e \in \Sigma_u \cup \Sigma_o$ and $t \in \{0, \ldots, n-1\}$, and
- ω^t for all $e \in \Sigma_u \cup \Sigma_o$, $\omega \in \delta(e)$, and $t \in \{0, \ldots, n-1\}$.

Next we describe $\mathcal{T}(t, t+1)$ for a given t which models the transition from time step t to time step $t+1$. When an event occurs, the event must be possible in the current state (1) and it has also some effects (2):

$$\omega^t \rightarrow \phi^t \quad \text{for every } \omega = \langle \phi, c\rangle \in \delta(e) \tag{1}$$

$$\omega^t \rightarrow \bigwedge_{l \in c} l^{t+1} \quad \text{for every } \omega = \langle \phi, c\rangle \in \delta(e) \tag{2}$$

The value of a state variable changes only if there is a reason for the change (*frame axiom*):

$$(a^t \wedge \neg a^{t+1}) \rightarrow (\omega_1^t \vee \ldots \omega_k^t) \tag{3}$$

for all $a \in A$ where $\omega_1 = \langle \phi_1, c_1\rangle, \ldots, \omega_k = \langle \phi_k, c_k\rangle$ are all event occurrences with $\neg a \in c_i$. For the change from *false* to *true* the formulae are defined similarly by interchanging a and $\neg a$.

An event can occur in only one way, and two events cannot be simultaneous if they interfere:

$$\begin{aligned}
&\neg(\omega_1^t \wedge \omega_2^t) \text{ for all } e \in \Sigma_o \cup \Sigma_u \text{ and } \{\omega_1, \omega_2\} \subseteq \delta(e) \\
&\qquad \text{such that } \omega_1 \neq \omega_2 \\
&\neg(\omega_1^t \wedge \omega_2^t) \text{ for all } \{e_1, e_2\} \subseteq \Sigma_o \cup \Sigma_u \text{ and } \omega_1 \in \delta(e_1) \text{ and} \\
&\qquad \omega_2 \in \delta(e_2) \text{ such that } \omega_1 \text{ and } \omega_2 \text{ interfere.}
\end{aligned} \tag{4}$$

The occurrence of events is represented by this formula:

$$(\bigvee_{\omega \in \delta(e)} \omega^t) \leftrightarrow e^t \text{ for all } e \in \Sigma_u \cup \Sigma_o \tag{5}$$

The conjunction of all the above formulae for a given t is denoted by $\mathcal{T}(t, t+1)$.[1] A formula for the initial state s_0 is:

$$\mathcal{I}_0 = \bigwedge_{a \in A | s_0(a)=1} a \wedge \bigwedge_{a \in A | s_0(a)=0} \neg a \tag{6}$$

A formula that represents all the behaviors described by the system for the length n is then

$$\Phi_{SD} = \mathcal{T}(0,1) \wedge \cdots \wedge \mathcal{T}(n-1, n) \wedge \mathcal{I}_0 \tag{7}$$

The observations are a set of timed occurrence of observable events: the observable event e occurred at time t iff $\langle e, d\rangle \in OBS$. More complex observations are considered in the next section. The representation of the observations consists of setting the e^t to *true* if this observable event $e \in \Sigma_o$ occurred at time step t, and to *false* if it did not.

$$\bigwedge_{\langle e,t\rangle \in OBS} e^t \wedge \bigwedge_{\langle e,t\rangle \notin OBS} \neg\, e^t \tag{8}$$

Theorem 1.
The solutions to the SAT problem $\Phi_{SD} \wedge \Phi_{OBS}$ represent the set of trajectories on SD consistent with the observations OBS and ending with the last observation of OBS.

To compute the diagnosis, we propose to *filter* the trajectories as proposed in (Torta & Torasso 2004) for static systems with the formula Φ_{Que} that is *true* if no faulty event occurred. Φ_{Que} is called the *query formula*.

$$\Phi_{Que} = \bigwedge_{e \in \Sigma_f, t \in \{0,\ldots,n-1\}} \neg e^t \tag{9}$$

The formula that finds sequences of events on the model that are consistent with the observations and that are normal is defined by: $\Phi_\Delta = \Phi_{SD} \wedge \Phi_{OBS} \wedge \Phi_{Que}$. If this formula is satisfiable, then the diagnosis of the system is normal. Otherwise, it is faulty.

[1] If a single event can occur at any time step t, the following clauses should be added: $\forall e_1, e_2 \in \Sigma_u \cup \Sigma_o$, $\neg e_1^t \vee \neg e_2^t$ or another equivalent representation of this property.

Dealing with uncertain observations

This section deals with non trivial observations. We first discuss the problem when the number n of time steps is unknown. We then consider different scenarios for the observations: total order, partial order, observations represented by an automaton.

Dealing with an unknown number of time steps

In the previous section, diagnosis is performed with the assumption that the time step of the occurrence of an observable event is known for each observation. It provides several obvious pieces of information (ordering of the observable events, time of occurrence of the observations) but also the number of time steps n in the trajectories consistent with the observations. How to translate the diagnosis problem if n is unknown? In planning by SAT, the goal is to find the shortest plan that enables to reach a goal state: the value of n is incremented until the SAT problem $\Phi(n)$ is satisfiable. In diagnosis, we have to consider *all* the trajectories in the system that are consistent with the observations. We now show that it is possible to restrict ourselves to some value n as equivalent diagnosis result is provided when only considering the trajectories of length n.

Let $\mathcal{E}_k = E_0, \ldots, E_{k-1}$ be a sequence of sets of events of length k on the model of the system, and let $OBS(\mathcal{E}_k)$ be the observations received by the diagnosis system after the occurrence of $\mathcal{E}_k$ ($\mathcal{E}_k$ may not determine $OBS(\mathcal{E}_k)$ uniquely). Let OBS be the observations actually received from the system. The diagnosis using the value n as the number of time steps is always correct if $\forall k \in \mathbf{N}$, $\forall \mathcal{E}_k = E_0, \ldots, E_{k-1}$ such that $OBS(\mathcal{E}_k) = OBS$, there exists a trajectory $\mathcal{E}_n = E_0, \ldots, E_{n-1}$ which is f-equivalent to $\mathcal{E}_k$ and such that $OBS(\mathcal{E}_n) = OBS$. There are different ways to use this result.

Let $\mathcal{E} = E_0, \ldots, E_{k-1}$ be a sequence of sets of events of length k in the system. Another sequence of sets of events of length $k+1$ in the system is $E_0, \ldots, E_{i-1}, \varepsilon, E_i, \ldots, E_{k-1}$. The latter is f-equivalent[2] to the former. This means that if there exists a normal/faulty trajectory of size k, there also exists a normal/faulty trajectory of size $k+1$. Thus, it is sufficient to use an upper bound of the total number of time steps. For instance, if the maximum number of unobservable events between two observable events is x and the number of observations emitted is p, then an upper bound of the number of events is $(x+1) \times p$.

However, such an upper bound may not exist since there may exist loops of unobservable events, or may be too high which leads to a huge number of propositional variables and a huge SAT problem. Fortunately, it is possible to consider a small value for x.

Let $\mathcal{E} = E_0, \ldots, E_{n-1}$ be the sequence of sets of events that occurred in the system and let i, j be two times of occurrence of consecutive observations such that $j - i$ is big. Since most events do not interfere, f-equivalent sequences of sets of events can be found by moving earlier or later some events between E_i and E_j, or by merging consecutive non interfering sets E_l and E_{l+1}. For instance, if E_i and E_{i+1} do not interfere $\mathcal{E}' = E_0, \ldots, E_i \cup E_{i+1}, \ldots, E_{n-1}$ is f-equivalent to $\mathcal{E}$ depending on the definition of a faulty behavior. Thus, it is sometimes possible to use a smaller value for x.

In the example presented in the second section, it is sufficient to consider that $x = 1$. Indeed for any trajectory, there is an equivalent trajectory such that the events between two observable events are all non interfering.

[2]Since the notion of f-equivalence mainly relies on the definition of normal/faulty behavior, one should be careful with this statement in case of non-trivial definitions of a faulty behavior.

Total order on the observations

The observations OBS are received in the order they were emitted, possibly immediately, but the number of unobservable events is unknown. OBS is a set of p pairs $\langle e, i\rangle$ where $e \in \Sigma_o$ and $i \in \{1, \ldots, p\}$ such that $\langle e, i\rangle \in OBS \wedge \langle e', i\rangle \in OBS \Rightarrow e = e'$. Let $o = \langle e, i\rangle \in OBS$ and $o' = \langle e', i'\rangle \in OBS$ be two observations, the event e of o occurred before e' of o' if $i < i'$.

The main issue is to find the occurrence time step of the observations. As seen in the previous subsection, we consider an upper bound of the number of unobservable events. We denote $d(i)$ the time step of occurrence of the ith observable event ($d(i) = (x+1) * i - 1$ if x is constant). The formula Φ_{OBS} is the conjunction of the following clauses:

$$\begin{array}{rl} \neg e^t & e \in \Sigma_o : \forall i,\ d(i) \neq t, \\ \neg e^{d(i)} & i \in \{1, \ldots, p\},\ e \in \Sigma_o : \langle e, i\rangle \notin OBS, \text{ and} \\ e^{d(i)} & i \in \{1, \ldots, p\},\ e \in \Sigma_o : \langle e, i\rangle \in OBS. \end{array}$$

Partial order on the observations

The observations are partially ordered. Let $o = \langle e, i\rangle \in OBS$ and $o' = \langle e', i'\rangle \in OBS$ be two observations. The partial order $\prec$ between the observations has the following semantics: $o \prec o'$ if the observable event e of o surely occurred before e' of o'.

As in the previous subsection, we denote $d(j)$ the time step of occurrence of the jth observable event in the sequence of observable event. Note that the observable event e of the observation $o = \langle e, i\rangle$ is not necessarly the ith event of the sequence. A propositional variable $o^{d(j)}$ is created that indicates that the observable event associated with o occurred at time step $d(j)$ and a variable $\hat{o}^{d(j)}$ that indicates that the observable event associated with o occurred *before* or at the time step $d(j)$.

The formula Φ_{OBS} is the conjunction of the clauses given in Table 1. An observable event occurred before $d(j)$ if it occurred before $d(j-1)$ or at $d(j)$ (10). An observation is emitted only once (11). All the observations were emitted (12). The partial ordering must be defined (13). Finally, an observable event occurs if and only if an observation associated with this event was received (14).

It is possible to reduce the number of propositional variables. If o is such that $o' \prec o$, then o cannot be the first observation emitted. Then, obviously $o^{d(1)} = \hat{o}^{d(1)} = \mathit{false}$ and it is not required to create these variables. If k observations o' are such that $o' \prec o$, then the variables $o^{d(1)}$ to $o^{d(k)}$

$$\hat{o}^{d(j)} \Leftrightarrow \hat{o}^{d(j-1)} \vee o^{d(j)} \quad (10)$$

$$\hat{o}^{d(j-1)} \Rightarrow \neg o^{d(j)} \quad (11)$$

$$\hat{o}^{d(p)} \qquad \forall o \in OBS \quad (12)$$

$$o_2^{d(j)} \Rightarrow \hat{o}_1^{d(j-1)} \qquad \forall o_1, o_2 : \; o_1 \prec o_2 \quad (13)$$

$$e^{d(j)} \Leftrightarrow o_1^{d(j)} \vee \cdots \vee o_k^{d(j)} \quad (14)$$

where $o_1, \ldots, o_k$ represent the observations emitted by the event e.

Table 1: Rules used to model the partial order

are useless. The same reasoning can be used for the successors of o. In the worst case, the number of variables required to represent the observations is quadratic in the number of observations. However, if for each observation o, there are at most k observations o' such that $o \not\prec o'$ and $o' \not\prec o$, then the number of variables is less than $2 \times k \times p$ (where p is the number of observations).

Observations represented as a finite state machine

Recent works on diagnosis consider uncertain observations: in the case of large distributed systems, the delay between the emission of an observation and its reception leads to a partial order on the observations; observations can be noisy; observations can be lost, etc. These uncertainties can be handled by representing the observations using a finite state machine, such as *index space* in (Lamperti & Zanella 2003), *observation automaton* in (Grastien, Cordier, & Largouët 2005), where each path from an initial state to a final state represents a possible sequence of observable events that occurred in the system.

The observations can be translated into a formula Φ_{OBS} in a similar way as for Φ_{SD}. Here, the main issue is to determine the number of time steps n. The modeling of the observations can even be generalised to the observations of some of the state variables.

Dealing with faulty behaviors

The diagnosis result in previous sections only indicates whether a faulty event occurred in the system. However, a more accurate result is generally expected, such as the number of faults that occurred or the identification of these faults. More recently, it has been proposed to consider *supervision patterns*.

Number of faults

We denote by Φ_{Que_0} the formula which indicates that the behavior of the system was correct. Given that $\Phi_{SD} \wedge \Phi_{OBS} \wedge \Phi_{Que_0}$ is not satisfiable, we know that at least one faulty event has occurred.

It is possible to build a formula Φ_{Que_i} that is satisfiable iff exactly i faults occurred in the system (Bailleux & Boufkhad 2003). The formula $\Phi_{\Delta_i} = \Phi_{SD} \wedge \Phi_{OBS} \wedge \Phi_{Que_i}$ is satisfiable if a scenario consistent with the observations contains i faults. As we mentioned earlier, faults can occur that cannot be diagnosed because not enough observations were received. For this reason, we are interested in minimal number of faults that possibly occurred on the system.

We propose to test the satisfiability of Φ_{Δ_i} starting from $i = 0$ and increasing the value of i until Φ_{Δ_i} is satisfiable. Then, the minimal number of fault that might have occurred is i.

Let j be the number of faults that occurred in the system. The SAT solver is called for $j + 1$ times. In case j is high, it is possible to multiply i by two at each step and test the satisfiability of $\Phi_{SD} \wedge \Phi_{OBS} \wedge (\Phi_{Que_i} \vee \cdots \vee \Phi_{Que_{i \times 2-1}})$ until a value i is found such that $i \leq j < i \times 2$ and then get the correct value of j by dichotomy. The SAT solver is then called for $2 \times \log_2 j$ times.

Localisation

We also want to identify the exact faults that have occurred. The diagnosis problem now becomes finding the set of all faults that have occurred in the system. Let i be the smallest value such that Φ_{Δ_i} is satisfiable. The solution provided by the SAT solver is an example of a faulty behavior in the system. The set of faulty events $S_1 = \{e_1^{t_1}, \ldots, e_i^{t_i}\}$ can be easily extracted from this solution.

S_1 may not be the only set of faults of size i that is consistent with the observations. If we are interested in obtaining all the minimal cardinality diagnoses, then it is possible to build a formula from Φ_{Δ_i} such that S_1 cannot be a solution:

$$\Phi_{\Delta_i} \wedge (\bigvee_{e^t \in S_1} \neg e^t).$$

If this formula is satisfiable, another diagnosis S_2 can be extracted and a new formula can be built that accepts neither S_1 nor S_2. If there exist k different diagnoses of size i, the SAT solver may be called $i + k + 1$ times.

Extended faulty behaviors

Jéron *et al.* (2006) have recently proposed to consider supervision pattern rather than a unique faulty event. A supervision pattern describes a set of system behaviors. In the previous sections, we considered the set of behaviors that contain a faulty event. In (Jéron *et al.* 2006), a supervision pattern is represented by a finite-state machine that can be easily translated into a formula Φ_{Que} in a similar way as for Φ_{SD} and Φ_{OBS}. More generally, we consider formulae Φ_{Que}, which contain information about the event occurrences and the state variables assignments.

Let QUE be a set of faulty patterns. Let $\{QUE_1, \ldots, QUE_k\}$ be a partition of QUE such that $\Phi \in QUE_i$ means that the *fault level* of the pattern Φ is i. If two patterns $\Phi \in QUE_i$ and $\Phi' \in QUE_j$ with $i < j$ are possible explanations of the observations, then the diagnosis should return the pattern Φ rather than Φ'. Define QUE_0 as the set of patterns which accept the behaviors not represented in QUE (i.e. the normal behaviors).

This is a generalisation of the previous subsection: the set of faulty patterns QUE_i contains all the faulty patterns where i faults occurred; moreover it is considered that there is a trajectory consistent with the observations and that contains i faults, then $j > i$ faults should not be diagnosed even

if there exists another trajectory consistent with the observations and that contains j faults.

The formula that is satisfiable if a faulty pattern of level i is consistent with the observation can be defined by:

$$\Phi_{\Delta_i} = \Phi_{SD} \wedge \Phi_{OBS} \wedge (\bigvee_{\Phi \in QUE_i} \Phi).$$

The satisfiability of Φ_{Δ_i} can be tested starting from $i = 0$ until Φ_{Δ_i} is satisfiable. Then, it is easy to extract the pattern $\Phi \in QUE_i$ that has been recognised. Equal level patterns can be found by testing the satisfiability of formula $\Phi_{\Delta_i} \wedge \neg\Phi_1 \wedge \cdots \wedge \neg\Phi_k$, where $\{\Phi_1, \ldots, \Phi_k\} \subseteq QUE_i$ are the patterns that have already been recognized, and extracting a new pattern Φ_{k+1} if the formula is satisfiable.

Experiments

Tests were performed on the system described at the beginning of the paper. The experiments were conducted on a 1.73 GHz Pentium 4 computer using state-of-the-art SAT solvers. We found that the best SAT solvers for this kind of SAT problems are the ones based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure (Davis, Logemann, & Loveland 1962) using the conflict-driven clause learning (CDCL) enhancement, such as Siege v4 (Ryan 2003), zChaff v151104, MINISAT v1.13, MINISAT v1.14, and MINISAT v2.0.[3] MINISAT 2.0 runs complete simplification procedure on input formula, in order to simplify the formula. Siege uses a seed, for the random number generator, taken from system time, so its performance is different for the same problem in different time. Thus, we run Siege 10 times on each problem for getting the minimum (MIN), maximum (MAX) and average (AVG) runtimes.

The first experiments were performed on a scenario with an increasing number of faults (from 1 to 20). The number of observations is about 10 times the number of faults. The observations are totally ordered and the goal is to determine the number of faults. The number of unobservable events between two observable events was set to $x = 1$. No incremental computation was used. Table 2 shows the runtime of the SAT solvers on the satisfiable formula $\Phi_{\Delta_k} = \Phi_{SD} \wedge \Phi_{OBS_k} \wedge \Phi_{Que_k}$; the results are also described graphically in Figure 2. For siege v4, we plot the average runtime of siege (siegeAVG) and describe its MIN and MAX runtimes by the grey region in the figure. The entries `#var` and `#cls` in the tables represent the number of variables and clauses in the CNF formula.

The table shows that SAT solvers solve the diagnosis problems very efficiently. Note that solving these problems by computing all consistent trajectories of the model using classical diagnosis approaches are hard. The Sampath's diagnoser approach would also fail because the diagnoser cannot be computed.

An important result is that the complexity of the computation does not necessarily increase with the size of the diagnosis window. For instance, MINISAT 2.0 takes 6.8s to solve

[3]These SAT solvers are available from http://www.satcompetition.org/.

	CNF properties		Runtime (s)			
k	`#var`	`#cls`	siege	zChaff	M 1.14	M 2.0
1	10 679	44 017	$0.01\ ^{0.01}_{\mathbf{0.01}}$	0.03	0.03	0.03
2	21 718	90 091	$0.03\ ^{0.03}_{\mathbf{0.03}}$	0.06	0.06	0.06
3	38 337	160 343	$0.09\ ^{0.09}_{\mathbf{0.1}}$	0.37	0.26	1.11
4	51 636	217 593	$0.29\ ^{0.13}_{0.53}$	**0.25**	0.42	0.73
5	70 415	299 361	$0.49\ ^{0.22}_{0.98}$	**0.67**	3.1	6.8
6	80 094	343 427	$0.54\ ^{0.26}_{\mathbf{1.2}}$	1.7	2	1.4
7	95 483	412 861	$2.4\ ^{1.1}_{5.2}$	7.1	33	**4.1**
8	111 032	484 053	$3.2\ ^{1.3}_{6.5}$	21	**3.8**	11
9	124 521	547 703	$2.6\ ^{1.2}_{\mathbf{6.0}}$	23	138	8.9
10	139 970	621 071	$6.7\ ^{1.1}_{30}$	9.6	**2.6**	27
11	153 604	687 472	$4.8\ ^{1.8}_{\mathbf{11}}$	15	21	18
12	171 108	772 361	$6.5\ ^{3}_{19}$	21	**7.2**	8.8
13	184 892	841 608	$5\ ^{2.5}_{\mathbf{7.0}}$	11	14	9.6
14	200 656	920 953	$4.7\ ^{2.8}_{6.4}$	18	**3**	14
15	214 585	992 951	$6.1\ ^{3.6}_{8.2}$	18	**5.4**	8.1
16	226 664	1 057 317	$6\ ^{4.2}_{8.9}$	16	**7**	7.7
17	244 261	1 149 035	$6.6\ ^{3.7}_{9.9}$	24	**4.8**	78
18	259 978	1 233 191	$9.1\ ^{5.2}_{18}$	32	7.7	**5.9**
19	275 735	1 318 745	$15\ ^{8.5}_{23}$	32	18	**8.1**
20	287 672	1 387 077	$10\ ^{4.3}_{23}$	20	11	**8.6**

Table 2: Runtime of SAT solvers on satisfiable Φ_{Δ_k} with totally ordered observations

Φ_{Δ_5} but only 1.4s to solve Φ_{Δ_6}. This relies on the property of the SAT solvers which do not perform a forward or a backward search but choose the variables in a dynamic order to get the smallest search tree: the behavior of Δ_k can be difficult to find, while it is obvious when new observations are provided in Δ_{k+1}. Interestingly enough, not all SAT solvers consider the same problem hard: Φ_{Δ_9} is harder than $\Phi_{\Delta_{10}}$ for MINISAT 1.14 (138s $>$ 2.6s), while it is easier for MINISAT 2.0 (8.9s $<$ 27s). The same tendancy can be found for Δ_9 and Δ_{17}. The issue of determining which algorithm is the more efficient and which heuristic should be used for this kind of SAT problem is an interesting open problem, particularly since the very similar MINISAT solvers give such different results.

In the diagnosis algorithm, Φ_{Δ_k} is computed by increasing k from $k = 0$ until Φ_{Δ_k} is satisfiable. In the planning domain, determining that there is no solution for $i = k - 1$ is often more expensive than finding a solution for $i = k$. Thus, we conduct a batch of experiments with $\Phi_{\Delta_k} = \Phi_{SD} \wedge \Phi_{OBS_k} \wedge \Phi_{Que_{k-1}}$ which is unsatisfiable. The runtime is limited to 600 seconds. The results are shown in Table 3.

The computation for these problems is much more expensive. The difficulty of these problems can be explained by the complexity associated with the computation of the num-

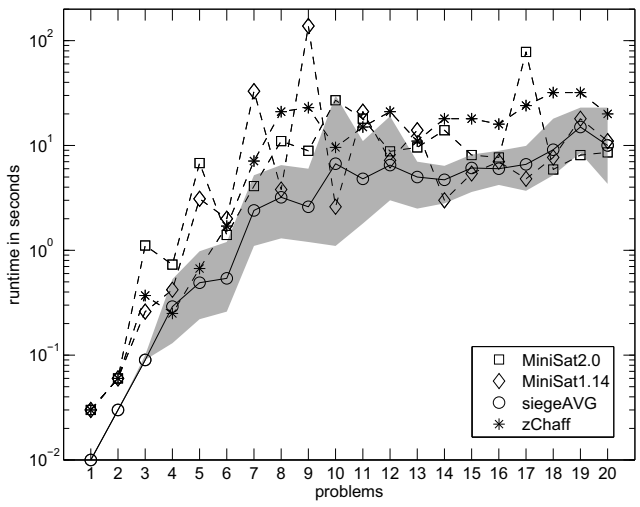


Figure 2: Runtime of SAT solvers on satisfiable Φ_{Δ_k} with totally ordered observations

	CNF properties		Runtime (s)			
k	#var	#cls	siege	zChaff	M 1.14	M 2.0
1	10 440	43 300	$0.01^{0.01}_{0.01}$	0.01	0.02	0.01
2	22 999	94 977	$0.02^{0.02}_{0.02}$	0.09	0.07	0.07
3	36 118	149 931	$0.1^{0.1}_{0.1}$	0.25	0.24	0.25
4	51 077	213 683	$0.54^{0.48}_{0.59}$	1.3	0.49	0.43
5	64 516	271 913	$1.4^{1.2}_{1.6}$	3	1.5	2
6	79 665	338 711	$7.4^{6.2}_{8.3}$	29	6	12
7	94 974	407 267	17^{12}_{23}	105	24	25
8	108 573	469 491	21^{16}_{28}	90	17	27
9	126 072	549 653	18^{13}_{25}	116	27	22
10	143 371	630 653	35^{23}_{47}	>600	62	41
11	155 090	688 191	39^{23}_{68}	>600	19	66
12	170 659	763 837	76^{68}_{84}	>600	52	128
13+	186 308	841 001	$>600^{>600}_{>600}$	>600	>600	>600

Table 3: Runtime of SAT solvers on unsatisfiable Φ_{Δ_k} with totally ordered observations

ber of faulty events. For instance, proving $\Phi_{SD} \wedge \Phi_{OBS_k} \wedge \Phi_{Que_0}$ is unsatisfiable only takes less than one second for any SAT solver used in our study for any given k.

The last test compares the runtimes when the observations become uncertain. We consider the following three cases: the observations are timed, totally ordered or partially ordered. When the observations are uncertain, an observation o_i was surely emitted before o_j, i.e $o_i \prec o_j$, iff $i + 5 < j$. The most efficient SAT solver for the partial order case being MINISAT 1.13, we present the result of this SAT solver in Table 4.

The computation is very simple in the case of timed observations. The runtime only increases slightly with new observations. However, the computation becomes very hard when the observations are partially ordered.

k	Timed observations	Total order	Partial order
1	0.02	0.03	0.04
2	0.07	0.08	0.09
3	0.16	0.26	0.14
4	0.36	0.3	3.6
5	2.6	1.9	9.7
6	3.5	16	5.6
7	4.8	9.1	350
8	4.1	24	290
9	4.3	61	63
10	4.2	25	>600
11	2.1	3.9	>600
12	2.7	6.3	>600
13	4.8	8.7	>600
14	4.3	16	>600
15	3.7	18.7	>600
16	11.8	9.6	>600
17	7.7	30	>600
18	9	13	>600
19	30	66	>600
20	12	16	>600

Table 4: Runtime of MINISAT 1.13 on satisfiable Φ_{Δ_k}

Discussion

Diagnosis with diagnosers (Sampath *et al.* 1995) is very efficient because processing an observation sequence can be done in linear time in the length of the sequence. This is in contrast to the SAT approach in which runtime in the worst case may grow exponentially in the length of the observation sequence. However, the construction of the diagnoser may be extremely expensive because the diagnoser may have a size that is exponential in the number of states in the system. In the SAT approach there is no similar direct dependency between the runtime and the number of states in the system. These two approaches represent a different tradeoff between on-line and off-line computation and between dependency of the number of states and the length of event sequences.

The other traditional approach computes the set of all trajectories and determines whether these trajectories are normal. A disadvantage is that the number of trajectories is often extremely high and it is not practical to compute all of them. Obviously, for diagnosing one given observation sequence most of the possible trajectories are not needed (and this is taken advantage of in the SAT approach) but, once the set of all trajectories has been computed, it can be used several times for different diagnosis queries. Recent works on this approach (Cordier & Grastien 2007) use decentralized or factored representations to represent the set of all trajectories more compactly without enumerating all of them.

The complexity of the SAT problem is exponential in the number of propositional variables. The number of propositional variables is linear with the number of state variables and rules, and linear in the number of timestep. This makes

the diagnosis by SAT a problem harder in general than the classical algorithms. However, as in planning by SAT, the structure in the SAT problem enables the SAT solvers to solve the problem efficiently.

Conclusion

We translated the diagnosis of discrete-event systems into a SAT problem and then used the state-of-the-art SAT solvers to solve this problem. Our results show that diagnosis problems, which are unreachable by the traditional diagnosis approaches, can be solved efficiently by SAT solvers. However, the diagnosis is still challenging when the number of observations increases. An answer to this problem would be incremental diagnosis approach, and this would be an interesting prospect for diagnosis by SAT. The difficulty is then to ensure that the diagnosis of a temporal window will be consistent with the next window.

The results also show that the existing SAT solvers have performance varying with no single best solver for any given formula. Moreover, the difference in runtime between solvers can be huge. Determining the best SAT algorithm for diagnosis problems is still an open question.

SAT algorithms efficiently explore the search space. We claim that traditional diagnosis algorithms can inspire from these algorithms to improve their efficiency.

Acknowledgements

This research was supported by NICTA in the framework of the SuperCom and the G12 projects. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian National Research Council.

References

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of Boolean cardinality constraints. In *Proc. of 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, 108–122.

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Proc. of 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-99)*, 193–207.

Cordier, M.-O., and Grastien, A. 2007. Exploiting independence in a decentralised and incremental approach of diagnosis. In *Proc. of 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 292–297.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communication of ACM* 5:394–397.

Grastien, A.; Cordier, M.-O.; and Largouët, Ch. 2005. Incremental diagnosis of discrete-event systems. In *Proc. of 16th International Workshop on Principles of Diagnosis (DX-05)*, 119–124.

Jéron, T.; Marchand, H.; Pinchinat, S.; and Cordier, M.-O. 2006. Supervision patterns in discrete-event systems diagnosis. In *Proc. of 17th International Workshop on Principles of Diagnosis (DX-06)*, 117–124.

Jiroveanu, G., and Boël, R. 2005. Petri net model-based distributed diagnosis for large interacting systems. In *Proc. of 16th International Workshop on Principles of Diagnosis (DX-05)*, 25–30.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. of 13th National Conference on Artificial Intelligence (AAAI-96)*, 1194–1201.

Lamperti, G., and Zanella, M. 2003. *Diagnosis of Active Systems*. Kluwer Academic Publishers.

Pencolé, Y., and Cordier, M.-O. 2005. A formal framework for the decentralised diagnosis of large scale discrete-event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)* 164:121–170.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)* 32:57–95.

Rintanen, J., and Grastien, A. 2007. Diagnosability testing with satisfiability algorithms. In *Proc. of 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 532–537.

Rintanen, J. 2007. Diagnosers and diagnosability of succinct transition systems. In *Proc. of 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 538–544.

Ryan, L. 2003. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Schumann, A.; Pencolé, Y.; and Thiébaux, S. 2007. A spectrum of symbolic on-line diagnosis approaches. In *Proc. of 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*.

Su, R., and Wonham, W. M. 2005. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *Transactions on Automatic Control* 50(12):1923–1935.

Torta, G., and Torasso, P. 2004. The role of OBDDs in controlling the complexity of model based diagnosis. In *Proc. of 15th International Workshop on Principles of Diagnosis (DX-04)*.

Veneris, A. 2003. Fault diagnosis and logic debugging using boolean satisfiability. In *Proc. of 4th International Workshop on Microprocessor Test and Verification: Common Challenges and Solutions*, 60–65.

Williams, B., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proc. of 13th National Conference on Artificial Intelligence (AAAI-96)*, 971–978.

Passive Robust Fault Detection: Inverse vs Direct Image Tests using Zonotopes

Pedro Guerra, Vicenç Puig, Ari Ingimundarson
Automatic Control Department
Universitat Politècnica de Catalunya (UPC)
Rambla de Sant Nebridi, 11, 08222 Terrassa (Spain)
vicenc.puig@upc.edu

Abstract

In this paper, a robust fault detection test based on calculating the inverse image of an interval model is presented. It relies on a consistency test which uses tools from interval analysis and zonotope arithmetic to check if there exists a member in the family of models described by an interval model that can explain the measured data. The proposed test is compared to the classical robust interval model fault detection approach based on the direct image of an interval function. The main features of both tests will be extracted and advantages and drawbacks discussed using a motivational example. Finally, an application example is given to assess how the algorithms work on a multivariable process.

Introduction

Model-based fault detection methods rely on the concept of *analytical redundancy*. The simplest analytical redundancy schema consists on comparing measurements of a system with corresponding analytically computed values that in turn are computed from measurements of other variables and/or from previous measurements of the same variable. The resulting differences are called *residuals* and are assumed to be indicative of faults in the system. Under ideal conditions, residuals are zero in the absence of faults and non-zero when a fault is present. When the residual becomes non-zero the relations used to calculate the residual are considered to be invalid. However, modelling errors and disturbances in complex engineering systems are inevitable, leading the residuals to non-zero values even in the absence of faults. Thus, the residual generation stage is followed by a decision-making stage. A robust fault detection system invalidates a residual relation only if model uncertainty and/or disturbances can not explain the observed data, see (Chen and Patton 1999).

Approaches to robust fault detection are divided in two principal groups. In the first, often referred to as *active* robust fault detection, robustness is achieved by generating residuals from which the effect of model errors and disturbances have been decoupled. This leads to residuals which are insensitive to uncertainty and disturbances but at the same time sensitive to faults. This approach has been extensively developed the last years (Chen and Patton 1999). The main drawback of this approach is the need of decoupling, being not always possible.

In the second approach, referred to as *passive*, the aim is to enhance the robustness of the fault detection system at the decision-making stage, mainly by using an adaptive threshold. A common approach to this problem is to inflate the allowable interval for the residual[1] so that false alarms due to model uncertainty are avoided. A limitation of this approach is that faults that produce a residual deviation smaller than the residual uncertainty due to model uncertainty will be missed.

A manner to represent model uncertainty is to bound parameter values on intervals. The resulting models are called *interval models* and have received a lot of attention in the context of robust fault detection, see among others (Armengol *et al.* 2001; Travé-Massuyés and Milne 1997; Puig *et al.* 2002; Escobet *et al.* 2001; Adrot *et al.* 1999; Ploix *et al.* 2000). Generally in these publications the uncertainty interval for residuals (or predicted outputs) is computed by propagating the effect of the parameter uncertainty using a *direct image* of an interval function. This approach will be referred to as a *direct image test* in what follows. The residual relation is invalidated if the variable for which the interval is calculated leaves the interval.

In this paper a method will be presented where the *inverse image* of interval functions is used to check wether there exists a member in the family of models, described by an interval model, that can explain the measured data. This inverse image test can be made very efficient using zonotope representation. This test will be compared with the existing direct image test and the principal properties of the two tests extracted. These properties will be demonstrated by using an example.

The paper is organized in the following manner: Section is dedicated to introducing the problem. In Sections and , the earlier direct image test and the inverse image test are described for comparison. In Section zonotopes and the related operations required for implementing fault detection tests are presented. A motivational example and an application example are presented in Sections and to demonstrate how the algorithms work. Finally in Section conclusions

[1] Often the intervals are calculated directly around process measurements instead of, in the residual case, around zero.

are drawn.

Problem set-up

Let us consider that the output of the monitored system can be described by

$$y(k) = \varphi^T(k)\theta(k) + e(k) \quad (1)$$
$$\theta(k+1) = \theta(k) + w(k) \quad (2)$$
$$\theta(k) \in \Theta \quad (3)$$

where $\theta(k) \in \mathbb{R}^n$ is the parameter vector whose values are assumed to be unknown but to belong to a compact bounded initial set Θ, $\varphi(k) \in \mathbb{R}^n$ is the regressor vector which can contain any function of inputs and outputs, the noise and parameter variations terms are limited as

$$|e(k)| \leq \sigma \qquad \text{and} \qquad |w(k)| \leq \lambda \quad (4)$$

respectively. As the parameter vector is assumed to belong to $\mathbb{R}^n$ so does λ and the last inequality is an element wise inequality. Notice that this system description includes any system linear in the parameters. Parameter uncertainty comes from physical modelling or from the set-membership parameter estimation algorithms applied in a non-faulty situation.

Notice that Eq. (2) specifies the allowed variance of uncertain parameters θ. Depending on the value of λ, three different cases can be considered:

- Time invariant case, $\lambda = 0$
- Time-varying case 1, $\lambda = \bar{\lambda}$
- Time-varying case 2, $\lambda = \infty$

In the first case, the parameter is unknown within Θ but it is known that it will not vary. In the second case, the parameter variation is bounded specifically by a vector $\bar{\lambda}$ while in the last case, the variation is implicitly bounded only by the initial parameter set Θ and can vary at will within that set.

The first case could represent situations when an initial variance comes from components specifications that are known only with a mean and variance in the beginning of the fault detection. The second case could represent a system that has been identified over a number of operation conditions, each with a different θ within Θ, but with the variance between samples bounded by $\bar{\lambda}$.

It is assumed that measurement data is available for $N+1$ ($\geq n$) points, that is, series

$$\Phi_N = \{\varphi(k)\}_{k=1,\ldots,N} \quad Y_N = \{y(k)\}_{k=1,\ldots,N} \quad (5)$$

are available at every time instant k.

Measurement noise can be taken into account by assuming that the measurements are known to belong to intervals $[y(k)]$, often created by adding an noise term $e(k)$ to the actual measurement $y(k)$, that is, $[y(k)] = [y(k)-e(k), y(k)+e(k)]$. The corresponding interval vector is

$$[Y_N(k)] = [y(k)] \times \cdots \times [y(k-N+1)] \subset \mathbb{R}^N$$

Fault detection using a direct image test

As already noted in the introduction, fault detection using interval models has predominantly been based on calculating the *direct image* of functions related to trajectory generation in order to obtain the set which $\hat{y}(k)$ belongs to

$$\hat{\mathbb{Y}}(k) = \{\hat{y}(k) \mid \hat{y}(k) = \varphi^T(k)\theta(k), \theta(k) \in \Theta\} \quad (6)$$

In fault detection using the *direct image test*, the model is assumed invalidated and fault is indicated if $Y_N(k) \notin \hat{\mathbb{Y}}_N$ or, in the case of measurement noise, if

$$[Y_N(k)] \cap \hat{\mathbb{Y}}_N = \emptyset \quad (7)$$

As the evaluation of the image in Eq. (7) is complex even in the linear case, approximate envelopes are often used based on computing the interval hull of $\hat{\mathbb{Y}}$, denoted as $\square\hat{\mathbb{Y}}$. Envelopes are characterized as *sound* and *complete* according to how well they approximate $\square\hat{\mathbb{Y}}$. A sound envelope does not contain any region through which a trajectory does not cross, that is the envelope $[\hat{Y}]$ fulfills $[\hat{Y}] \subseteq \square\hat{\mathbb{Y}}$. A complete envelope $[\hat{Y}]$ on the other hand fulfills $\square\hat{\mathbb{Y}} \subseteq [\hat{Y}]$.

A general form of the fault detection algorithm using the direct image test is the following:

Algorithm 1 Fault detection using the direct image test

1: $fault \leftarrow FALSE$
2: $k \leftarrow 1$
3: **while** fault=FALSE **do**
4: Obtain input-output data $\{u(k), y(k)\}$ at time instant k and build regressor $\varphi(k)$.
5: Calculate $\hat{\mathbb{Y}}(k)$ as a direct image of Θ according to Eq. (6)
6: **if** $\hat{\mathbb{Y}}(k) \cap [y(k)] = \emptyset$
7: $fault \leftarrow TRUE$
8: **endif**
9: $k \leftarrow k + 1$
10: **end while**

It is known that it could be difficult to detect some faulty trajectories with the direct image test even when the envelope is sound and complete (Armengol *et al.* 2001). This is due to the fact that many combinations of parameters can explain a set of data, a jump between these combinations will not be detected. On the other hand, using the direct image test it is difficult to consider explicitly the parameter variations presented in Eq. (4).

Fault detection using a inverse image test

Intuitive idea

An alternative fault detection test to check the consistency of the model given by Eqs. (1)-(3) will now be proposed which is based on the inverse image of the function in Eq. (1). Assume the measurement data $y(k)$ is available. Then, the interval $[y(k)]$ created by taking into account measurement noise $e(k)$. Intuitively, what is proposed now is a test based on computing the *inverse image* of $[y(k)]$, i.e.,

$$\hat{\Theta}(k) = f^{-1}[y(k)] \quad (8)$$

where $\hat{\Theta}(k)$ is the set of parameters consistent with the measurement data at time instant k. In this case, the model or parameter set Θ is invalidated if

$$\Theta \cap \hat{\Theta}(k) = \emptyset \tag{9}$$

A failed test means that there does not exist a $\theta \in \Theta$ that is consistent with the measurements.

Formalized idea

In order to formalize the inverse image test, the following definitions are introduced.

Definition 1. Let us consider the model description given by Eq. (1), for given data sequences Φ_N and Y_N introduced in 5, the parameter θ is said to belong to the *Feasible Solution Set* at time N, (denoted FSS_N), if there exist $\theta(1), \theta(2), \ldots, \theta(N)$ such that:

- $\theta = \theta(N)$ (10)
- $|y(k) - \varphi^T(k)\theta(k)| \leq \sigma \quad k = 1, \ldots, N$ (11)
- $|\theta(k) - \theta(k-1)| \leq \lambda \quad k = 2, \ldots, N$ (12)
- $\theta(k) \in \Theta \quad k = 1, \ldots, N$ (13)

Using previous definition, a fault is now defined for the sequences Φ_N and Y_N.

Definition 2. For given data sequences Φ_N and Y_N, a *fault* is said to have occurred if the set FSS_N is empty.

Each new measurement defines a set of consistent parameters defined by

$$F_k = \{\theta \in \mathbb{R}^n : -\sigma \leq y(k) - \varphi(k)^T\theta \leq \sigma\} \tag{14}$$

F_k is the region between two hyperplanes. The normalized form of this strip is written as

$$\begin{aligned} F_k &= \{\theta \in \mathbb{R}^n : |\frac{y(k)}{\sigma} - \frac{\varphi(k)}{\sigma}^T \theta| \leq 1\} \\ &= \{\theta \in \mathbb{R}^n : |d(k) - c(k)^T\theta| \leq 1\} \end{aligned} \tag{15}$$

This strip F_k available at time k allows to iteratively detect the presence of a fault if its intersection with the feasible parameter set FSS_k is empty.

In practice, the computation of FSS_N is difficult. The fault detection algorithm presented in this paper is based on using zonotopes as an approximated feasible solution set, AFSS_N, that fulfills $\mathrm{FSS}_N \subseteq \mathrm{AFSS}_N$ for which consistency is checked. In the case when $\lambda > 0$, the set AFSS_N is expanded to take the allowed parameter variance into account in the next sample. The expanded set is denoted $\overline{\mathrm{AFSS}}_{N+1}$.

A general form of the suggested fault detection algorithm is the following:

Algorithm 2 Fault detection using the inverse image test

1: $fault \leftarrow FALSE$
2: $k \leftarrow 1$
3: $\overline{AFSS}_k \leftarrow \Theta$
4: **while** fault=FALSE **do**
5: Obtain input-output data $\{u(k), y(k)\}$ at time instant k, build regressor $\varphi(k)$ and strip F_k according to Eq. (15).
6: **if** $\overline{AFSS}_k \cap F_k = \emptyset$
7: $fault \leftarrow TRUE$
8: **else** Calculate AFSS_k that fulfills $F_k \cap \overline{\mathrm{AFSS}}_k \subset \mathrm{AFSS}_k$ and
9: Expand AFSS_k taking into account λ to obtain $\overline{\mathrm{AFSS}}_{k+1}$.
10: **endif**
11: $k \leftarrow k + 1$
12: **end while**

Zonotopes and related operations

Zonotopes

In this section, zonotopes and related operations will be presented as a tool to implement *Algorithm 1* and *Algorithm 2*.

Definition 3. *The **Minkowski sum** of two sets $\mathbb{X}$ and $\mathbb{Y}$ is defined by $\mathbb{X} \oplus \mathbb{Y} = x + y : x \in \mathbb{X}, y \in \mathbb{Y}\}$.*

Definition 4. *Given a vector $p \in \mathbb{R}^n$ and a matrix H $\in \mathbb{R}^{n \times m}$, the Minkowski sum of the segments defined by the columns of matrix H, is called a **zonotope** of order m and it is represented as (see Fig. 1):*

$$\mathbb{X} = p \oplus HB^m = \{p + Hz : z \in B^m\}$$

where: B^m is a unitary box, composed by m unitary intervals.

The order m is a measure for the geometrical complexity of the zonotopes.

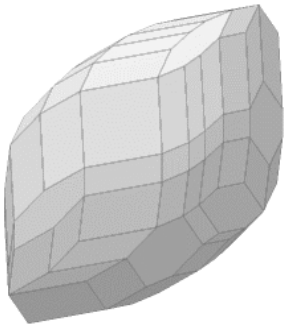


Figure 1: Zonotope of order m=14

Looking at *Algorithm 1*, the required zonotope-based operations are: the direct image of a zonotope through a linear transformation (step 5) and the intersection between zonotopes (step 6). In case of *Algorithm 2*, the following zonotope-based operations are required: checking the consistency of a zonotope with a strip (step 6), intersection between a zonotope and a strip (step 8) and the expansion of the parameter set taking into account λ (step 9).

Image of a zonotope through a linear transformation

Consider a zonotope represented by $\mathbb{X} = p \oplus HB^m$ where $p \in \mathbb{R}^n$ is a vector and $H \in \mathbb{R}^{n \times m}$ is a matrix. The image

of a zonotope through a linear transformation $M \in \mathbb{R}^{n\times n}$ is a zonotope $\mathbb{Y}$ defined by:

$$\mathbb{Y} = q \oplus NB^m \tag{16}$$

where: $q = Mp$ and $N = MH$.

Intersection between two zonotopes

Given two zonotopes $\mathbb{X}_1 = p_1 \oplus H_1B^{r_1}$ and $\mathbb{X}_2 = p_2 \oplus H_2B^{r_2}$ and matrix E, let us define:

$$\widehat{p}(E) = Ep_1 + (I-E)p_2 \tag{17}$$

$$\widehat{H}(E) = [EH_1 \;\; (I-E)H_2] \tag{18}$$

then,

$$\mathbb{X}_1 \cap \mathbb{X}_2 \subseteq \widehat{\mathbb{X}}(E) \tag{19}$$

$$\widehat{\mathbb{X}}(E) = \widehat{p}(E) \oplus \widehat{H}(E)B^{r_1+r_2} \tag{20}$$

To reduce the size of the intersection zonotope $\widehat{\mathbb{X}}(E)$, a convex optimization problem is solved. If H_{1i} and H_{2j} (with *i*=1,$\cdots$,m_1, *j*=1,$\cdots$,m_2) are the columns of matrices H_1 and H_2, the function to be minimized is:

$$f(E) = \sum_{i=1}^{m_1}(EH_{1i})^T(EH_{1i}) + \sum_{j=1}^{m_2}(H_{2j}-EH_{2j})^T(H_{2j}-EH_{2j}) \tag{21}$$

Checking consistency of a zonotope with a strip

Given a new data point $\{y(k)\}$ at time instant k, regressor $\varphi(k)$ and strip F_k according to Eq. (15) are build. Assuming that $FSS_k \subseteq \mathbb{X}$ where $\mathbb{X} = p \oplus HB^m$ is a zonotope, consistency can be assessed by checking if

$$\mathbb{X} \cap F_k = \emptyset \tag{22}$$

This check is very easy to perform using the following definition:

Definition 5. *A hyperplane $\mathcal{S} = \{x : c^Tx = q\}$ is a* ***supporting hyperplane*** *of a zonotope $\mathbb{X} = p \oplus HB^m$ if either $c^Tx \le q_u, \forall x \in \mathbb{X}$ or else $c^Tx \ge q_d, \forall x \in \mathbb{X}$ with equality occurring for some $x \in \mathbb{X}$. The two constants q_u and q_d characterizing the supporting hyperplanes are easily calculated as*

$$q_u = c^Tp + \|H^Tc\|_1 \tag{23}$$

$$q_d = c^Tp - \|H^Tc\|_1 \tag{24}$$

where $\|\cdot\|_1$ is the 1-norm of a vector.

Then, calculating the supporting hyperplane constant q_u and q_d the intersection is empty if and only if

$$q_u < \frac{y(k)}{\sigma} - 1 \quad \text{or} \quad q_d > \frac{y(k)}{\sigma} + 1 \tag{25}$$

This condition of inconsistency was reported in (Vicino and Zappa 1996).

Intersection between a zonotope and a strip

Given the zonotope $\mathbb{X} = p_k \oplus HB^r$, the tight strip $\mathbb{S} = \{\theta \in \mathbb{R}^n : |c^T\theta - d| \le \sigma\}$, then for every integer j,$0 \le j \le r$

$$\mathbb{X} \cap \mathbb{S} \subseteq \Theta = \widehat{v}(j) \oplus T(j)B^r \tag{26}$$

where:

It is possible to choose the parameter vector α in such a way that a size criterion for the obtained bound is minimized. Here we use the method based in the Frobenius norm proposed in (Alamo *et al.* 2005).

Expansion of the parameter set

The bound on parameter variation can be expressed as $|\theta(k+1) - \theta(k)| < \Lambda$ which in turn can be expressed as

$$\theta(k+1) \in \theta(k) \oplus \Lambda B^n \tag{27}$$

where Λ is a square matrix with the diagonal equal to λ. One of the principal features of zonotopes is that the Minkowski sum of a box and a zonotope is another zonotope. Therefore, if at time k it is known that the parameter belongs to set $\mathbb{X}_k = p \oplus HB^m$ then using Eq. (27) the parameter set at time $k+1$ can be expressed as $\mathbb{X}_{k+1} = p \oplus HB^m \oplus \Lambda B^n = p \oplus [H\ \Lambda]B^{m+n}$.

Notice that in this step the zonotope order increases at each time instant. In order to control the domain complexity, a reduction step is thus implemented. Here we use the method proposed in (Combastel 2003) to reduce the zonotope complexity:

Property 1. *Given the zonotope $\mathbb{X} = p \oplus HB^r \subset \mathbb{R}^n$ and the integer s (with $n < s < r$), denote by $\widehat{H}$ the matrix resulting from reordering the columns of matrix H in decreasing Euclidean norm. Then $\mathbb{X} \subseteq p \oplus \left[\widehat{H}_T \;\; Q\right] B^s$, where $\widehat{H}_T$ is obtained from the first s-n columns of matrix $\widehat{H}$ and $Q \in R^{n\times n}$ is a diagonal matrix that satisfies: $Q_{i,i} = \sum_{j=s-n+1}^{r} |\widehat{H}_{ij}|$, i=1...,n.*

Remark. In the time-varying case ($\lambda = \infty$) as the parameters can vary at will within the initial parameter set Θ, the update procedure of step 9 of *Algorithm 2*, consists on resetting $\overline{AFSS}_{k+1}$ equal to the initial parameter set Θ.

Motivational example

In order to compare the fault detection tests based on the direct and inverse image, a simple motivational example will be used.

Consider a static system with two inputs, $y(k) = au_1(k) + bu_2(k)$. Assume that $a \in [0.5, 0.65]$ and $b \in [0.4, 0.55]$. The output is corrupted by measurement noise, $e(k)$ known to be bounded on $|e(k)| \le 0.2$.

Direct image test

Using the direct image test, a data point $\{y(k), u_1(k), u_2(k)\}$ generated with parameters a and b will be rejected if

$$y(k) > \max_{(x,y)\in[a]\times[b]} xu_1(k) + yu_2(k)$$

$$y(k) < \min_{(x,y)\in[a]\times[b]} xu_1(k) + yu_2(k)$$

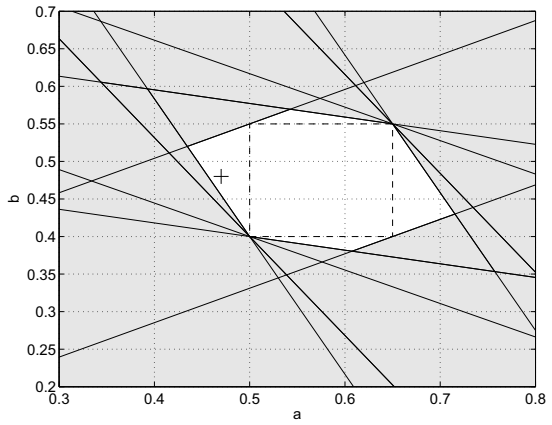


Figure 2: Parameter box, bands of allowable parameters and faulty point +, direct image case. The gray area is excluded by intersection of the bands.

as $y(k)$ does not belong to the direct image of $[a] \times [b]$ given $u_1(k)$ and $u_2(k)$. For each data point $\{y(k), u_1(k), u_2(k)\}$ these inequalities define a band of parameters that would not have been rejected by that data point. In Fig. 2 the intersection of such bands for 5 data points is shown in white. The set of parameters that would have been rejected by these data points is shown in grey. For the 5 data points $y(k)$ was calculated using parameters $(a, b) = (0.47, 0.48)$ which are outside the allowed parameter set and displayed with a cross in the figure.

A number of observations will now be made. The region outside the box but inside the intersection of the bands is the set of parameters that are outside the allowable set but would not have been detected with the direct image test. The size of this set depends on two things, the "excitation" of the data and the size of the intersection of the bands.

Regarding the excitation of the data some extreme cases can be pointed out. To design a perfect detection test, excitation would be a combination of two data points where one input is zero at a time. This way any point outside the allowable parameter set would have been detected. To detect parameter deviations close to the original allowable set, one of the input signals needs to approach zero. Notice that four of the bands touch the allowed parameter set at points (0.5,0.4) and (0.65,0.55) while only one touches at points (0.65,0.4) and (0.5,0.55). The reason for this was that only one of the data point had $u_1(k)$ and $u_2(k)$ with different signs. If all $u_1(k)$ and $u_2(k)$ would have had the same sign the test would have been much less restrictive as then none of the bands would have touched the box at points (0.65,0.4) and (0.5,0.55). The data points were generated with a faulty parameter. It is clear that if one of the data points would have the correct excitation, the fault would have been detected.

The size of the intersection of the bands depends on the width of the allowed parameter set $[\theta]$ as the bands are calculated using the corner points of $[\theta]$.

Finally, if the parameters would have varied between data points while staying in the allowed parameter set, this variation would not have been detected.

The reason for these characteristics of the direct image test is the loss of dependence between parameters and the outputs while computing the envelope. Only maximum and minimum values of the parameters are used for the envelope. This leads to tests that are specially sensitive to faults close to the extreme points of the parameter intervals (corners) while less sensitive to faults where some parameters are close to the center of the interval and where many more parameter combinations can explain the data. Moreover, parameter variance inside $[\theta]$ is not detected because even thought parameters change, the measured trajectory stays within the direct image $[\hat{Y}]$.

Inverse image test

In this case, a point (a, b) will be rejected if it fulfills one of the following two conditions,

$$y(k) + e(k) < au_1(k) + bu_2(k) \quad (28)$$
$$y(k) - e(k) > au_1(k) + bu_2(k) \quad (29)$$

This again defines a band of parameters consistent with each data point. In Fig. 3, the intersection of the bands for the same 5 data points used in the previous sections are shown. Again the grey area are parameters that have been rejected while the white area is the set of parameters consistent with the data. Since the intersection of the white area with the allowed parameter box is empty the fault is detected. It is also clear that if the parameter would have changed between data points this would have caused the corresponding bands to be translated causing the intersection of all bands to be empty.

Regarding the size of the intersection of the band, it is much smaller than in the direct image case. Its size depends primarily on $e(k)$ and the combination of $\{y, u_1, u_2\}$. Excitation matters less and the condition to detect a fault close to the original parameter set depends again more on the measurement noise $e(k)$.

Application example

A quadruple-tank process (see (Johansson and R. Nunes 1998)) is proposed as the application example to further compare the two fault detection tests proposed.

A diagram of the process is shown in figure 4. The process inputs are v_1 and v_2 (input voltages to the pumps). The continuous model derived from the mass balances and the Bernoulli's law is:

$$\frac{d\,h_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1 \quad (30)$$
$$\frac{d\,h_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2 \quad (31)$$
$$\frac{d\,h_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}v_2 \quad (32)$$
$$\frac{d\,h_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}v_1 \quad (33)$$

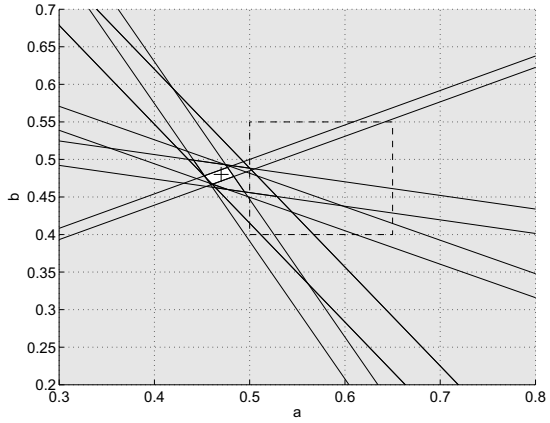


Figure 3: Parameter box, bands of allowable parameters and faulty point $+$, inverse image case. The gray area is excluded by intersection of the bands.

where A_i is the cross section of the tank i and a_i is the cross section of the outlet hole of the tank i. The vector of measured outputs is composed by the levels of four tanks, denoted h_i. The parameters $\gamma_1, \gamma_2 \in (0, 1)$ determine the flow to different tanks, that is, the flow to tank 1 is $\gamma_1 k_1 v_1$ and the flow to tank 4 is $(1 - \gamma_1)k_1 v_1$ and similarly for tank 2 and tank 3.

The fixed values used in the simulations are $A_1 = A_3 = 28 \quad cm^2$, $A_2 = A_4 = 32 \quad cm^2$, $k_1 = 3.33 \quad cm^3/Vs$, $k_2 = 3.35 \quad cm^3/Vs$ and $g = 981 \quad cm/s^2$. To obtain a system of the form given by Eq. (1) to use the fault detection procedures proposed in this paper, an euler discretization with step size equal to 1 of one equation of the model is used:

$$\begin{aligned} h_1(k+1) &= h_1(k) + 1 * (-\frac{a_1}{A_1}\sqrt{2gh_1(k)} \\ &+ \frac{a_3}{A_1}\sqrt{2gh_3(k)} + \frac{\gamma_1 k_1}{A_1} v_1(k)) \\ &+ e_1(k) \end{aligned} \quad (34)$$

where $|e_i(k)| \leq 0.02$ with $i = 1, 2$ is a bounded random noise. Note that the results obtained in this section could be improved using the other three equations of the model. Then, the parameter vector θ is composed of: $\theta = [a_1 \;\; a_2 \;\; a_3 \;\; a_4 \;\; \gamma_1 \;\; \gamma_2]^T$. The regressor vector used corresponding to the first output $y_1 = h_1$ is:

$$\varphi_{y_1}(k) = \left[-\frac{\sqrt{2gh_1(k)}}{A_1} \;\; 0 \;\; \frac{\sqrt{2gh_3(k)}}{A_1} \;\; 0 \;\; \frac{k_1 v_1(k)}{A_1} \;\; 0 \right]^T$$

In this paper, parameters a_1 and γ_1 are assumed to belong to the intervals $a_1 \in [-0.029, 0.171]$ and $\gamma_1 \in [0.55, 0.85]$. To compare the direct image and inverse image fault detection approaches, three different cases of parametrical fault scenarios were studied, in each one occurs a jump on parameters a_1 and γ_1. For all cases the non-faulty system is simulated with parameters equal to $a_1 = a_3 = 0.071 \quad cm^2$, $a_2 = a_4 = 0.057 \quad cm^2$, $\gamma_1 = 0.7$ and $\gamma_2 = 6$. The initial parameter uncertainty set for the fault detection stage is assumed to be:

$$\Theta = \{\theta : \theta = p_0 + H_0\tilde{\theta}, \|\tilde{\theta}\|_\infty \leq 1\}$$

where: $p_0 = [0.071 \;\; 0.057 \;\; 0.071 \;\; 0.057 \;\; 0.7 \;\; 0.6]^T$, $H_0 = diag\left([0.1 \;\; 0 \;\; 0 \;\; 0 \;\; 0.15 \;\; 0]^T\right)$.

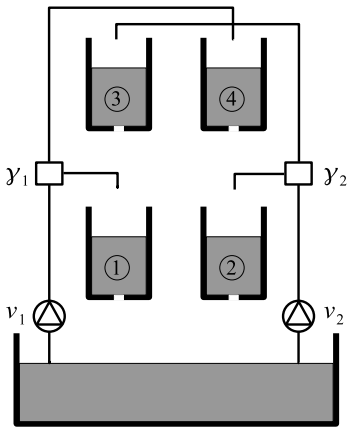


Figure 4: Quadruple-tank process

Time-invariant parameters

Since uncertain parameters are considered time-invariant, then $\lambda = 0$ in Eq. (4). The fault considered is a variation in the parameters a_1 and γ_1 from time instant $k = 5$. This variation of parameters is inside the allowed interval of both parameters, that is $a_{1f} = a_1 + 0.05$ and $\gamma_{1f} = \gamma_1 + 0.1$. Figure 5 shows the fault detection test using the inverse image. The dashed box represents the allowable parameters a_1 and γ_1. The solid line represents the zonotope intersection of the parameters consistent with the first 4 measurement outputs. The dotted line represents the band of parameters consistent with the measurement output for time instant $k = 5$. As this band does not intersect with the zonotope, a fault is indicated. Figure 6 shows the envelopes generated by the direct image fault detection test and the measurement output $[h_1]$ considering the measurement noise. As the measurement output never leaves the envelopes, no fault is indicated. This shows how the inverse image test is able to detect a fault that consists on a non-allowed time variation of the parameters, while the direct image test is unable.

Time-varying parameters case 1

In this case, uncertain parameters are considered time-varying with: $\lambda = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.03 \end{bmatrix}$, in Eq. (4). The fault considered is a variation in the parameters a_1 and γ_1 from time instant $k = 5$: $a_{1f} = a_1 + 0.03$ and $\gamma_{1f} = \gamma_1 + 0.05$. Even though with this variation, the parameters are inside their uncertainty intervals, it is higher than the allowed value

at each time instant given by λ. Figure 7 shows the fault detection test using the inverse image. The dashed box represents the valid interval for parameters a_1 and γ_1. The solid lines represent the zonotope that bounds the parameters consistent with the first 4 measurement outputs. The dotted line represents the band of parameters consistent with the measurement output for time instant $k = 5$. As this band does not intersect with the zonotope, a fault is indicated. Figure 8 shows the envelopes generated by the direct image fault detection test and the measurement output $[h_1]$ considering the measurement noise. As the measurement output never leaves the envelops, no fault is indicated. As in the previous case, this shows how the inverse image test is able to detect a fault that consists on a non-allowed time variation of the parameters, while the direct image test is unable.

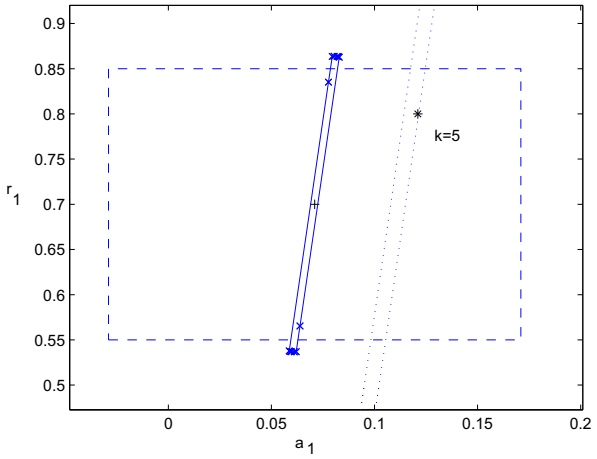


Figure 5: Inverse image test for time invariant case, a_1 vs. γ_1

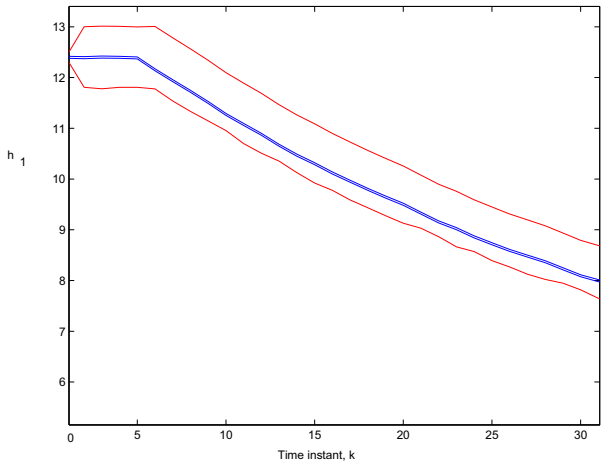


Figure 6: Direct image test for time invariant case

Time-varying parameters case 2

In this case, uncertain parameters are considered time-varying with $\lambda = \infty$, in Eq. (4). This means that variation is bounded only by the initial parameter set Θ, varying at will within this set. The fault considered is outside the box of allowed parameters, from time instant $k = 5$, that is $a_{1f} = a_1 + 0.15$. Figure 9 shows the fault detection test using the inverse image. The dashed box represents the allowable parameters region for a_1 and γ_1. The dotted line represents the band of parameters consistent with the measurement output for time instant $k = 5$. As this band does not intersect with the box of allowed parameters, a fault is indicated. Figure 10 shows the envelopes generated by the direct image fault detection test and the measurement output $[h_1]$ considering the measurement noise. As the measurement output leaves the envelopes from time instant $k = 6$, a fault is indicated. In this case, both methods detect the fault, being not clear the advantage of using the inverse image test.

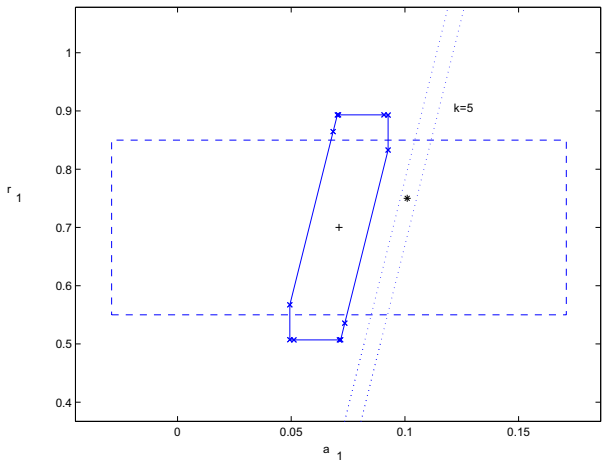


Figure 7: Inverse image test for time-varying case 1, a_1 vs. γ_1

Acknowledgments

This work has been funded by the grant CICYT DPI2006-11944 of Spanish Ministry of Education and by Research Comission of the Generalitat de Catalunya (group SAC ref. 2005SGR00537).

Conclusions

A robust fault detection test based on the inverse image using zonotopes for systems linear in the parameters has been introduced. A general algorithm was presented based on proving that the feasible solution set of parameters for a series of data is empty. Three distinct cases of allowed parameter variance have been considered to compare the inverse test with the traditional interval based fault detection test based on the direct image. Finally, both methods were applied to motivational example and to a simulation model of a multivariable process, showing the effectiveness of the inverse

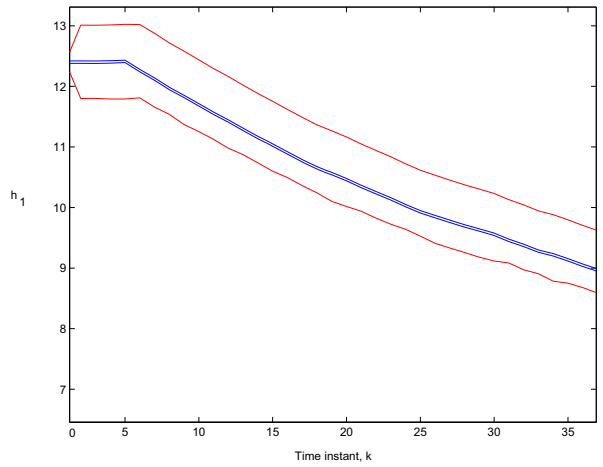


Figure 8: Direct image test for time-varying case 1

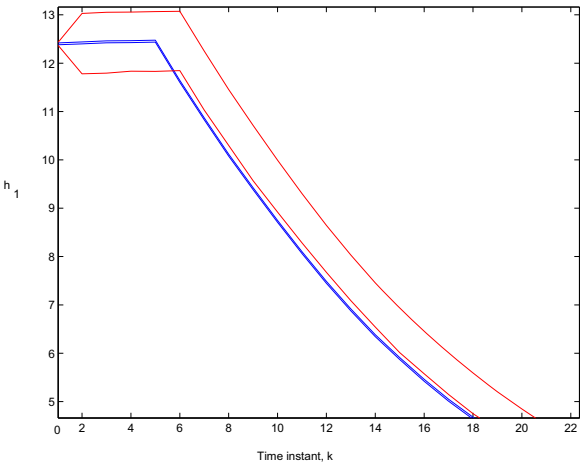


Figure 10: Direct image test for time-varying case 2

image fault detection test when considering time varying and invariant parameters.

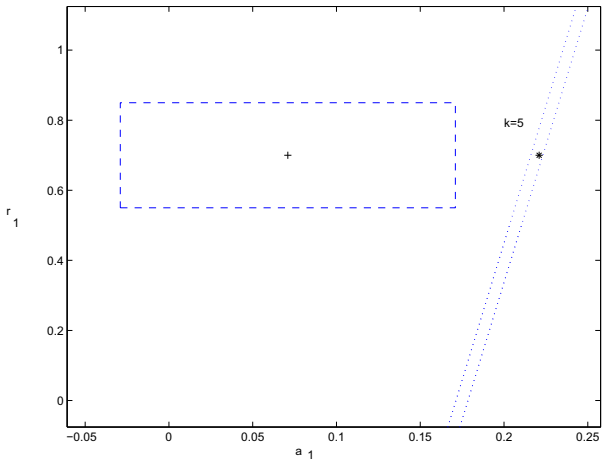


Figure 9: Inverse image test for time-varying case 2, a_1 vs. γ_1

References

O. Adrot, D.Maquin, and J. Ragot. Fault detection with model parameter structured uncertainties. In *European Control Conference (ECC'99)*, Germany, 1999.

T. Alamo, J.M. Bravo, and E.F. Camacho. Guaranteed state estimation by zonotopes. *Automatica*, 41(6):1035–1043, 2005.

J. Armengol, J. Vehi, L. Travé-Massuyés, and M. A. Sainz. Application of multiple sliding time windows to fault detection based on interval models. In *in Proc. 12th Int. Workshop on Principles of Diagnosis (DX-01)*, page 916, Sansicario, Italy, 2001.

J. Chen and R.J. Patton. *Robust Model-Based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers, 1999.

C. Combastel. A state bounding observer based on zonotopes. In *European Control Conference (ECC'03)*, Cambridge, UK, 2003.

T. Escobet, L.Travé-Massuyés, S. Tornil, and J. Quevedo. Fault detection of a gas turbine fuel actuator based on qualitative causal models. In *European Control Conference (ECC'01)*, Portugal, 2001.

Karl Henrik Johansson and J. L. R. Nunes. A multivariable laboratory process with an adjustable zero. In *Proc. 17th American Control Conference*, Philadelphia, Pennsylvania, 1998.

S. Ploix, O. Adrot, and J. Ragot. Bounding approaches to the diagnosis of uncertain static systems. In *IFAC SAFEPROCESS'00*, Hungary, 2000.

V. Puig, J. Quevedo, and T. Escobet. Robust fault detection approaches using interval models. In *IFAC World Congress (b'02)*, Spain, 2002.

L. Travé-Massuyés and R. Milne. Tiger: Gas turbine condition monitoring using qualitative model based diagnosis. *IEEE Expert Intelligent Systems and Applications*, 1997.

A. Vicino and G. Zappa. Sequential approximation of feasible parameter sets for identification with set membership unvertainty. *IEEE Transactions on Automatic Control*, 41:774–785, 1996.

On Monotonic Monitoring of Discrete-Event Systems

Gianfranco Lamperti and **Marina Zanella**

Dipartimento di Elettronica per l'Automazione
Università di Brescia
Via Branze 38, 25123 Brescia, Italy

Abstract

Two diagnosis tasks can be applied to discrete-event systems: a-posteriori diagnosis and monitoring-based diagnosis. The former is carried out under the assumption that all the observable events emitted by the system within the time interval of interest have already been received. In monitoring-based diagnosis, instead, the observation is received fragment by fragment and, in the general case, such an assumption does not hold, that is, some observable events generated by the system in the interval between two consecutive observation fragments may not be received within the current fragment, while they will be received in subsequent ones. Moreover, the relative emission order of some observed events may not be inferable from the observation, which, in such a case, is temporally uncertain. The diagnostic engine, however, is supposed to output a set of candidate diagnoses at the reception of each fragment. A problem arises about the significance of monitoring results: two consecutive set of diagnoses may be unrelated to one another. Even worse, a set of diagnoses may include no candidate relevant to the actual diagnosis. To cope with this problem, the notion of monotonic monitoring is introduced, which is supported by specific constraints on the fragmentation of the observation, leading to the notion of a stratified observation. The paper shows that a sound and complete diagnostic engine can achieve monotonic monitoring if a stratified observation is provided.

Introduction

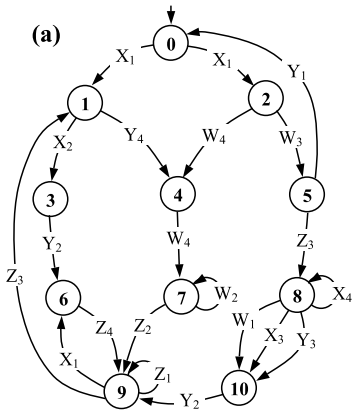
Model-based diagnosis of discrete-event systems (DESs) has arisen a great interest in the last decade (Rozé 1997; Debouk, Lafortune, & Teneketzis 2000; Lunze 2000; Console, Picardi, & Ribaudo 2002; Pencolé & Cordier 2005). All approaches in the literature adopt compositional modeling, that is, a DES consists of several interconnected components, where the favorite formalism to represent the behavior of each component is an automaton. Interconnections between components can either be modeled through explicit *ad hoc* primitives (Baroni *et al.* 1999) and/or implicitly, that is, a communication buffer may be indistinguishable from a component (Pencolé & Cordier 2005). Distinct approaches to diagnosis of DESs in the literature can either assume that several state changes of distinct components can occur simultaneously (Sampath *et al.* 1996; Su & Wonham 2005) or not. All diagnosis tasks requires an observation as input. Therefore, observation features and models have been investigated (Lamperti & Zanella 2000; 2002; Grastien, Cordier, & Largouët 2005). Basically, an observation is *temporally uncertain* if the generation order of observed events is not precisely known, that is, what is known is a partial order which conforms to the actual generation order. In other words, an event can be observed before another that was generated by the DES before it, and, given the reception order of events, it is impossible to devise the relative emission order of all the pairs of events belonging to the observation. Therefore, several sequences of observable events comply with a temporally uncertain observation.

Two diagnostic tasks inherent to DESs can be singled out: a-posteriori diagnosis (Lamperti & Zanella 2006) and monitoring-based diagnosis (Sampath, Lafortune, & Teneketzis 1998; Rozé & Cordier 2002; Lamperti & Zanella 2004). The former finds out the faults affecting a DES in an off-line way with respect to the system, by typically processing the whole observation gathered before the evolution of the system came to a halt. The latter, instead, tries and follow the evolution of a DES while it is occurring.

The claim of this paper is that the criterion of soundness and completeness of results is not meaningful for monitoring-based diagnosis in case the considered observation is affected by temporal uncertainty since it does not guarantee an important property that we call *monotonicity*. Such a property consists in producing as output at each monitoring step a set of diagnoses that includes the actual diagnosis. This paper discusses how monotonicity depends not only on the abilities of the problem-solving method but also on the characteristics of the considered observation, and its contribution is to explicit the constraints under which a temporally uncertain observation has to be considered by a sound and complete problem-solving method so as diagnostic results are monotonic, whichever the DES at hand.

Discrete-event systems

All the possible evolutions over time of a DES (we cumulatively call them the *global system behavior*) can be thought of as the paths of a directed graph, where each node is a *system state* (this being the composition of the states of all components and explicit links, if any) and each arc is a system state change, called a *transition*. This is a conceptual



(b)

	1	2	3	4
X	a	x		
Y		b		y
Z		z	c	
W			w	d

Figure 1: Behavior space $Bhv(\bar{\Sigma}, \bar{\Sigma}_0)$ (a), and viewer and ruler matrix (b) for $\bar{\mathcal{V}}$ and $\bar{\mathcal{R}}$.

standpoint, not an operational one, in that most current approaches in the literature need not generating any global system behavior. In other words, approaches to diagnosis of DESs can reason about all the feasible evolutions of a DES without generating them, by just exploiting the individual behaviors of components and connections, the assumptions about the (either simultaneous or not) state change triggering, and (possibly) also domain dependent constraints. The reason for we take the global system behavior into consideration is that is establishes a common ground for defining the outputs of any diagnosis task inherent to DESs, independently of any specific (modeling and processing) approach.

Formally, given a system Σ and an initial state Σ_0, each evolution of Σ is confined within the *behavior space*, $Bhv(\Sigma, \Sigma_0)$. The latter is a directed graph rooted in Σ_0, where each node is a state of Σ and each arc is a transition. Each path within the behavior space is a *history* of Σ. As such, a history h is a (possibly empty) sequence of transitions rooted in Σ_0.

Example 1. Fig. 1(a) draws the behavior space of a DES called $\bar{\Sigma}$. Nodes represent system states, where 0 is the initial state. Each arc represents a state change and is marked by a transition identifier. A path rooted in 0 is a possible history of $\bar{\Sigma}$, for instance, $\bar{h} = \langle X_1, X_2, Y_2, Z_4, Z_3, Y_4, W_4, Z_2, X_1 \rangle$.

Since the adopted conceptual representation of the global system behavior is quite general and sharable by distinct approaches to model-based diagnosis of DESs (and, therefore, distinct classes of DESs) in the literature, it is pointless to distinguish whether each transition is an individual component state change or not. For instance, X_i, Y_i, W_i, and Z_i, $i \in [1\,..\,4]$, might be the completely asynchronous transitions of a distributed DES featuring four components X, Y, W, and Z. Or X_i, Y_i, and W_i might be the asynchronous transitions of three components while each transition Z_i might be triggered simultaneously in the three components altogether. In either case what is dealt with in the next sections is the same. □

Diagnosis

A-posteriori diagnosis finds out the faults affecting a DES, given a relevant observation and the initial state of the system. The system, starting from its initial state, is assumed to have undergone a sequence of state transitions, some of which are associated with an observable event and/or a fault. We call *actual history* the sequence of transitions actually followed by the system, and *actual diagnosis* the set of faults entailed by the actual history. The output of the a-posteriori diagnosis task is a set of candidate diagnoses. A diagnosis is a *sound* candidate if it is entailed by a history that generates a sequence of observable events that comply with the given observation. The set of histories inherent to an observation is complete if it encompasses all and only the paths, included in the global system behavior, that start from the given initial state and produce a sequence of observable events that comply with the observation. The complete set of histories entails the *complete* set of candidate diagnoses.

Formally, the observability and abnormality properties of a DES can be represented as follows. Let $\mathbf{T}$ be the domain of transitions in Σ and $\mathbf{L}_o$ a domain of *observable labels*. A *viewer* of Σ is a function from $\mathbf{T}$ to $(\mathbf{L}_o \cup \{\epsilon\})$, where ϵ is the *null* label. If $(T, \epsilon) \in \mathcal{V}$ then T is *silent* else T is *visible*. $\mathcal{V}$ is said to cause *source uncertainty* when it includes two pairs (T_1, ℓ) and (T_2, ℓ) where $T_1 \neq T_2$. Let h be a history of Σ, the *signature* $h \boxtimes \mathcal{V}$ is the sequence of observable labels relevant to h,

$$h \boxtimes \mathcal{V} = \langle \ell \mid T \in h, (T, \ell) \in \mathcal{V}, \ell \neq \epsilon \rangle. \quad (1)$$

Example 2. Assuming $\mathbf{L}_o = \{a, b, c, d\}$, a viewer $\bar{\mathcal{V}}$ for $\bar{\Sigma}$ is defined by the following associations, displayed in the white cells of the matrix of Fig. 1(b), with the other transitions being silent: (X_1, a), (Y_2, b), (Z_3, c), (W_4, d). Based on $\bar{\mathcal{V}}$, the signature of history $\bar{h}$ defined in Example 1 is $\bar{h} \boxtimes \bar{\mathcal{V}} = \langle a, b, c, d, a \rangle$. □

Ideally, the signature should represent how h is observed outside Σ. However, what is actually perceived is a *relaxation* $\mathcal{O}$ of $h \boxtimes \mathcal{V}$, called the *observation* of Σ,

$$\mathcal{O} = \mathcal{U}(h \boxtimes \mathcal{V}). \quad (2)$$

$\mathcal{U}$ is a (nondeterministic unknown) *uncertainty function* that generates a DAG, $\mathcal{O} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of states and $\mathcal{A}$ the set of arcs, with the following *uncertainty properties*:

- (*Logical uncertainty*) Each label ℓ in the signature is perceived as a subset of $(\mathbf{L}_o \cup \{\epsilon\})$ of *candidate labels*, necessarily including ℓ;
- (*Temporal uncertainty*) Absolute temporal ordering of the signature is relaxed to partial ordering (with the latter being consistent with the former).

The uncertainty function is not given once and for all systems; instead it depends on the specific setting of the system at hand (sensors, channels conveying sensor values to the observer, etc.) Since such a function is nondeterministic, distinct instances of the same system run may be perceived by the same observer as different observations, represented by distinct DAGs. Let N be a node in $\mathcal{N}$. The *extension* of

N, written $\|N\|$, is the set of labels embodied in N. A *candidate signature* of $\mathcal{O}$ is a sequence of labels in $\mathbf{L}_o$ obtained by picking up a label from each $\|N\|$, $N \in \mathcal{N}$, without violating the ordering imposed by $\mathcal{A}$. The *extension* of $\mathcal{O}$, $\|\mathcal{O}\|$, is the whole set of candidate signatures of $\mathcal{O}$. If $\mathcal{O}$ is the observation relevant to a history h, then the signature relevant to h is among the candidate signatures of $\mathcal{O}$, as claimed by Proposition 1.

Proposition 1. *If* $\mathcal{O} = \mathcal{U}(h \boxtimes \mathcal{V})$ *then* $(h \boxtimes \mathcal{V}) \in \|\mathcal{O}\|$.

Proof. Let $\mathcal{O} = (\mathcal{N}, \mathcal{A})$. Let $h_v = \langle T_1, T_2, .., T_n \rangle$ be the sequence of visible transitions in h, that is, $h_v = \langle T_i \in h \mid (T_i, \ell_i) \in \mathcal{V}, \ell_i \neq \epsilon \rangle$. Therefore, $\mathcal{N}$ consists of n nodes. Let $p = \langle N_1, N_2, .., N_n \rangle$ be the (only) permutation of all the n nodes in $\mathcal{N}$ that has the absolute temporal ordering of the signature of h. Hence, there exists a bijective mapping between h_v and p, according to which T_i corresponds to N_i and vice versa. According to the definition of temporal uncertainty, the ordering of p is compatible with $\mathcal{A}$. Now we generate a candidate signature s of $\mathcal{O}$ by picking up from each $N_i \in \mathcal{N}$, without violating the ordering imposed by p, the label ℓ_i generated by T_i. Note that, according to the definition of logical uncertainty, $\ell_i \in \|N_i\|$. Hence s equals the signature of h, that is, $s = h \boxtimes \mathcal{V}$, and, at the same time, being a candidate signature, it belongs to the extension of $\mathcal{O}$, that is, $s \in \|\mathcal{O}\|$, which proves the proposition. □

Example 3. Based on $\bar{h}$ and $\bar{\mathcal{V}}$, depicted in Fig. 2 is an observation $\bar{\mathcal{O}} = \mathcal{U}(\bar{h} \boxtimes \bar{\mathcal{V}})$ of $\bar{\Sigma}$. Note how logical uncertainty holds in node N_4, where $\|N_4\| = \{d, \epsilon\}$, and temporal uncertainty involves nodes N_2 and N_3, whose reciprocal emission order is unknown. The extension of $\bar{\mathcal{O}}$ is[1] $\|\bar{\mathcal{O}}\| = \{abca, acba, abcda, acbda\}$ which, in accordance with Proposition 1, includes $\bar{h} \boxtimes \bar{\mathcal{V}} = abcda$. □

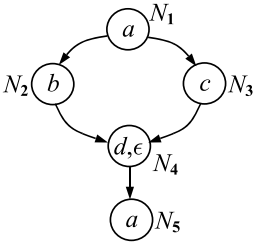


Figure 2: Observation $\bar{\mathcal{O}}$ for $\bar{\Sigma}$.

Like a viewer, we can define a mapping $\mathcal{R}$, the *ruler* of Σ, from $\mathbf{T}$ to $(\mathbf{L}_f \cup \{\epsilon\})$, where $\mathbf{L}_f$ is a set of *fault labels*. If $(T, \epsilon) \in \mathcal{R}$ then T is *normal* else T is *faulty*. Given $\mathcal{R}$, the *diagnosis* $h \otimes \mathcal{R}$ is the set of fault labels

$$h \otimes \mathcal{R} = \{\ell \mid T \in h, (T, \ell) \in \mathcal{R}, \ell \neq \epsilon\}. \quad (3)$$

Note how a diagnosis is empty when all transitions in h are normal.

Example 4. Assuming $\mathbf{L}_f = \{x, y, z, w\}$, a ruler $\bar{\mathcal{R}}$ for $\bar{\Sigma}$ is defined by the following associations displayed in the gray cells of the matrix of Fig. 1(b), with the other transitions being normal: (X_2, x), (Y_4, y), (Z_2, z), (W_3, w). Based on $\bar{\mathcal{R}}$, the diagnosis of history $\bar{h}$ defined in Example 1 is $\bar{\delta} = \bar{h} \otimes \bar{\mathcal{R}} = \{x, y, z\}$. □

An *a-posteriori diagnostic problem* relevant to a system Σ is a 4-tuple involving an initial state, a viewer, an observation, and a ruler,

$$\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R}). \quad (4)$$

Intuitively, the *solution* of $\wp(\Sigma)$, written $\Delta(\wp(\Sigma))$, consists of a set of *candidate diagnoses*, with each diagnosis being entailed by a history h in $Bhv(\Sigma, \Sigma_0)$ whose signature conforms to $\mathcal{O}$. As such, $\Delta(\wp(\Sigma))$ is

$$\{\delta \mid \delta = h \otimes \mathcal{R}, h \in Bhv(\Sigma, \Sigma_0), h \boxtimes \mathcal{V} \in \|\mathcal{O}\|\}. \quad (5)$$

The set of candidate diagnoses defined in (5) is sound and complete; in addition, it includes the diagnosis relevant to the actual evolution of Σ, as claimed by Proposition 2.

Proposition 2. *Let* $\wp(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ *be a diagnostic problem, where* $\mathcal{O} = \mathcal{U}(h \boxtimes \mathcal{V})$ *and* $\delta = h \otimes \mathcal{R}$. *Then,* $\delta \in \Delta(\wp(\Sigma))$.

Proof. By the definition of behavior space, the actual history h of the system that caused the diagnostic problem $\wp(\Sigma)$ belongs to $Bhv(\Sigma, \Sigma_0)$. Owing to Proposition 1, its signature belongs to the extension of $\mathcal{O}$, that is, $h \boxtimes \mathcal{V} \in \|\mathcal{O}\|$. Therefore, owing to the definition of the set of candidate diagnoses provided by Eq. (5), $\Delta(\wp(\Sigma))$ includes diagnosis δ, where, as assumed $\delta = h \otimes \mathcal{R}$. □

If h is the (unknown) actual history of Σ, then $\Delta(\wp(\Sigma))$ includes (besides the relevant diagnosis δ) a (possibly empty) set of *spurious diagnoses*, each of which is entailed by (at least) one history $h' \neq h$ such that $(h' \boxtimes \mathcal{V}) \in \|\mathcal{O}\|$.

Example 5. Let $\wp(\bar{\Sigma}) = (\Sigma_0, \bar{\mathcal{V}}, \bar{\mathcal{O}}, \bar{\mathcal{R}})$. Drawn in Fig. 1(b) is a matrix representing both viewer $\bar{\mathcal{V}}$ and ruler $\bar{\mathcal{R}}$: for each pair (C, i), $C \in \{X, Y, Z, W\}$, $i \in [1 .. 4]$, a (either observable or faulty) label is inserted[2]. This allows us to associate labels with the arcs of the behavior space. Note how the solution of $\wp(\bar{\Sigma})$ includes the diagnosis relevant to history $\bar{h}$, namely $\bar{\delta} = \{x, y, z\}$. □

Monitoring

Monitoring-based diagnosis finds out the faults affecting a DES iteratively, once for each considered chunk of observation, as soon as such a chunk has been received. Therefore, solving a monitoring-based diagnosis problem can be seen as repeatedly solving a-posteriori diagnosis problems, where the a-posteriori diagnosis problem solved at time $t + 1$ is inherent to the observation considered at time t plus the chunk of observation received in the meantime from t to $t + 1$. However, from the operational point of view, the methods for solving the two classes of problems are different since in monitoring-based diagnosis the problem at time $t + 1$ is solved by exploiting the solution of the problem at time t.

[1] For convenience, signatures are written as strings of labels.

[2] Since, in the example, faulty transitions are (incidentally) silent, each matrix element contains one label at most, where faulty labels are shaded.

The quality of the results produced by an a-posteriori diagnosis method is usually assessed based on their soundness and completeness. In fact, if all the outputs are actually candidates and no candidate is missing, this criterion guarantees that the (only) actual diagnosis is included in the set of candidates provided as output. This conclusion, however, holds only under the condition that the observation is complete, that is, all the observable events inherent to the considered time interval have already been received. Such a condition, in general, is not fulfilled during monitoring, wherein some observable events included in the observation chunk received by time $t+1$ may have been emitted before some events included in the previously received chunks.

The uncertain observation taken as input by an a-posteriori diagnosis task can still be represented as a DAG. However, this DAG is not given as a whole but, rather, as a list of several *fragments*, where each fragment is composed of one or several nodes along with the relevant temporal constraints (arcs).

Formally, let $\mathcal{O} = (\mathcal{N}, \mathcal{A})$. A *fragmentation* of $\mathcal{O}$ is a sequence

$$\mathcal{O}^* = \langle F_1, \ldots, F_n \rangle \tag{6}$$

where each fragment $F_i = (\mathcal{N}_i, \mathcal{A}_i)$, $i \in [1\,..\,n]$, is composed of a subset of nodes $\mathcal{N}_i \subseteq \mathcal{N}$ and a subset of arcs $\mathcal{A}_i \subseteq \mathcal{A}$ such that $\{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ and $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ are partitions of $\mathcal{N}$ and $\mathcal{A}$, respectively. Each fragment F_i represents a set of observable events received in the current time interval. Each node in $\mathcal{N}_i$ is an event and each arc in $\mathcal{A}_i$ is a temporal precedence relationship (according to the emission order). In particular, $\mathcal{A}_i$ is the minimal set of precedence relationships linking the observable events received in the current interval with each other and with the other events belonging to the observation. In fact, precedence is a transitive relationship and, therefore, if the event represented by $N_a \in \mathcal{N}_i$, according to an arc in $\mathcal{A}_i$, is preceded by an event represented by another (parent) node N_b, then all the predecessors of N_b are (implicit) ancestors of N_a, without any need of mentioning them in $\mathcal{A}_i$. In other words, $\mathcal{A}_i$ includes all and only the relationships linking the nodes in $\mathcal{N}_i$ with their parent nodes.

That given above is the most general definition of a fragmented observation, since it does not constrain the set of parent nodes of a given node in the observation DAG. This means that an event received in the current interval may have one or more parent events (i.e. events that precede it in the emission order) that have not been received yet.

Such a freedom degree can be suppressed by imposing the following condition, which requires the parents of nodes in the new fragment to be in the fragments received up to now:

$$\forall N \mapsto N' \in \mathcal{A}_i \left(N' \in \mathcal{N}_i, N \in \bigcup_{j=1}^{i} \mathcal{N}_j \right). \tag{7}$$

Each nonempty prefix $\langle F_1, \ldots, F_i \rangle$ of $\mathcal{O}^*$ corresponds to a *sub-observation* $\mathcal{O}_{[i]} = (\mathcal{N}_{[i]}, \mathcal{A}_{[i]})$, where

$$\mathcal{N}_{[i]} = \bigcup_{j=1}^{i} \mathcal{N}_j \qquad \mathcal{A}_{[i]} = \bigcup_{j=1}^{i} \mathcal{A}_j \tag{8}$$

In practice, if $\mathcal{O}$ is known, $\mathcal{O}^*$ is univocally defined by the sequence of $\mathcal{N}_i$, as (7) entails that $\mathcal{A}_i$ necessarily includes all (and only) the arcs entering nodes in $\mathcal{N}_i$.

We define the *empty* sub-observation $\mathcal{O}_{[0]} = (\emptyset, \emptyset)$. Note how, for each $i \in [0\,..\,n]$, we can define a *sub-problem* $\wp_{[i]}(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}_{[i]}, \mathcal{R})$.

Example 6. A fragmentation $\bar{\mathcal{O}}^*$, where $\bar{\mathcal{O}}$ is displayed in Fig. 2, is defined by the following sequence of sets of nodes: $\langle \{N_1, N_3\}, \{N_2\}, \{N_4\}, \{N_5\} \rangle$. □

A *monitoring problem* is a 4-tuple involving an initial state, a viewer, a fragmented observation, and a ruler,

$$\mu(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}^*, \mathcal{R}). \tag{9}$$

The *solution* of $\mu(\Sigma)$ is the sequence of the solutions of the diagnostic sub-problems $\wp_{[i]}(\Sigma)$, $i \in [0\,..\,n]$,

$$\Delta(\mu(\Sigma)) = \left\langle \Delta(\wp_{[0]}(\Sigma)), \ldots, \Delta(\wp_{[n]}(\Sigma)) \right\rangle. \tag{10}$$

Example 7. Let $\mu(\bar{\Sigma}) = (\Sigma_0, \bar{\mathcal{V}}, \bar{\mathcal{O}}^*, \bar{\mathcal{R}})$ be a monitoring problem inherent to the evolution described in Example 1. Then, $\Delta(\mu(\bar{\Sigma})) = \langle \Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4 \rangle$, where $\Delta_0 = \{\emptyset\}$, $\Delta_1 = \{\{w\}\}$, $\Delta_2 = \{\{w\}, \{x\}, \{x, y\}\}$, $\Delta_3 = \{\{w\}, \{x\}, \{x, y\}, \{x, y, z\}\}$, and $\Delta_4 = \{\{w\}, \{x, y, z\}\}$. □

Example 7 shows that the solution of a monitoring problem, although consisting in a sound and complete set of diagnoses, is quite disappointing. In fact, at monitoring step 1, one is induced to believe that w is a quite certain fault but, from iteration 2 on, fault w is not certain any more. The rationale behind this deceitful behavior is that any sound and complete reasoning mechanism produces a set of outputs that comply with the whole observation received so far as it were a complete observation, while it is not. Since, at any new step, events can be received that may precede (and/or follow) those belonging to the observation received up to the previous step, the extension of the observation may change non-monotonically from one step to another, thus producing the highlighted negative effect.

Example 8. In Example 7 the histories consistent with the (only) candidate signature ac of $\mathcal{O}_{[1]}$ are those belonging to the graph displayed in Fig. 3(a), where the final states are double circled. The histories consistent with $\mathcal{O}_{[2]}$, displayed in Fig. 3(b), are those producing either the candidate signatures acb or abc. Fig. 3(b) includes a left subgraph that causes two new diagnoses to be added in Δ_2 with respect to Δ_1. This subgraph has been completely added since no observation prefix of abc had been considered in the previous step. This is a consequence of the fact that nodes N_2 and N_3, whose reciprocal temporal order is unknown, have been considered in two different steps since they belong to two distinct fragments.

Note that a candidate diagnosis output at a certain time step, even if the only one, may be completely refuted in subsequent steps. This would have occurred, for instance, to candidate diagnosis $\{w\}$ at the second step of our example if there were no histories in the global behavior space producing the sequence of observable events acb.

□

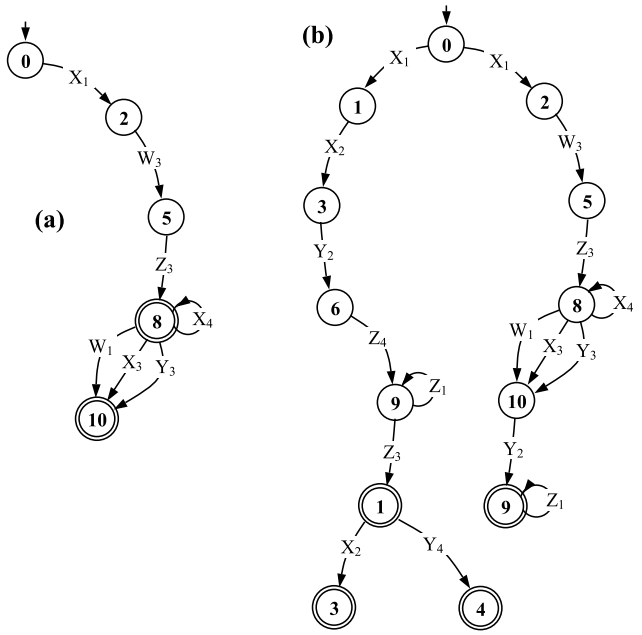


Figure 3: Nonmonotonic update of histories.

The negative effects we have highlighted are not a consequence of the restriction imposed on the definition of a fragmented observation by Condition (7). In fact, if such a condition does not hold, the effects are amplified. Therefore, we keep on considering the notion of a fragmented observation that fulfills Condition (7).

Stratification

Given a monitoring-based diagnosis context, we call *monotonicity* the property of producing as output at each iteration a set of candidates that includes the actual candidate. Monotonicity is a very important property in monitoring: results that are not monotonic are hardly useful and dependable.

Formally, let $\mu(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}^*, \mathcal{R})$, where $\mathcal{O}^* = \langle F_1, \ldots, F_n \rangle$. Let $\langle \Delta_0, \Delta_1, \ldots, \Delta_n \rangle$ be the solution of $\mu(\Sigma)$ and δ the actual (unknown) diagnosis of the (unknown) history of Σ. Since $\Delta_n = \Delta(\wp_{[n]}(\Sigma)) = \Delta(\wp(\Sigma))$, based on Proposition 2, we have $\delta \in \Delta_n$. We say that $\mu(\Sigma)$ is *monotonic* iff $\forall i \in [0\,..\,(n-1)]$ there exists $\delta_i \in \Delta_i$ such that

$$\delta_0 \subseteq \delta_1 \subseteq \ldots \subseteq \delta_{n-1} \subseteq \delta. \tag{11}$$

Example 9. The monitoring problem $\mu(\bar{\Sigma})$ in Example 7 is *not* monotonic: the actual diagnosis is $\bar{\delta} = \{x, y, z\}$, for which Condition (11) does not hold as $\Delta_1 = \{\{w\}\}$ includes no diagnosis that is a subset of $\bar{\delta}$. □

Interestingly, the monotonicity of a monitoring problem $\mu(\Sigma)$ depends on the nature of the fragmentation of $\mathcal{O}$. On the one hand, not all fragmentations of $\mathcal{O}^*$ make $\mu(\Sigma)$ monotonic. On the other, the *trivial fragmentation* involving the whole observation $\mathcal{O}$ as the unique fragment supports monotonicity, but this is in fact a-posteriori diagnosis, not monitoring. Thus, we are interested in the nature of nontrivial fragmentations that guarantee monotonicity, independently of the specific system (and behavior space) at hand, namely nontrivial *stratified observations*. The formal definition of a stratified observation requires the introduction of three functions applied on a node N of $\mathcal{O}$:

- $Anc(N)$: the set of *ancestor nodes* of N;
- $Desc(N)$: the set of *descendant nodes* of N;
- $Unrl(N) = \mathcal{N} - (Anc(N) \cup Desc(N))$: the set of *unrelated nodes* of N, i.e. all the nodes whose reciprocal emission order with respect to N is unknown.

A fragmentation $\mathcal{O}^* = \langle F_1, \ldots, F_n \rangle$ is *stratified* iff for each fragment $F_i = (\mathcal{N}_i, \mathcal{A}_i)$, $i \in [1\,..\,n]$, we have

$$\forall N \in \mathcal{N}_i \;\; (Unrl(N) \subseteq \mathcal{N}_i)\,. \tag{12}$$

A stratified fragmentation is called a *stratification* and each fragment a *stratum*. Condition (12) requires that all nodes, which are neither ancestors nor descendants (namely, unrelated) of nodes in the i-th stratum, be in the i-th stratum themselves. This allows candidate signatures of sub-observations to grow incrementally (monotonically), as claimed by Proposition 3.

Proposition 3. *Let $\mathcal{O}^* = \langle F_1, \ldots, F_n \rangle$ be a stratified observation. Then, for each $i \in [1\,..\,n]$, $\|\mathcal{O}_{[i]}\|$ is composed of all the signatures in $\|\mathcal{O}_{[i-1]}\|$ (possibly) extended with further observable labels.*

Proof. If the fragmentation is trivial, $\mathcal{O}^* = \mathcal{O}_{[1]} = \mathcal{O} = \langle F_1 \rangle$, then $\|\mathcal{O}_{[1]}\|$ includes all the candidate signatures of $\mathcal{O}$. Since, for every fragmented observation $\|\mathcal{O}_{[0]}\| = \{\epsilon\}$, and each signature in $\|\mathcal{O}_{[1]}\|$ either equals or extends the null signature, the proposition is proved for $n = 1$ (and $i = 1$).

If $n > 1$, let $F_i = (\mathcal{N}_i, \mathcal{A}_i)$, $i \in [1\,..\,n]$. Let $\|F_i\|$ be the *stratum extension* of F_i, which is the set of all the *signature postfixes* of F_i. A signature postfix of a stratum F_i is a sequence of labels in $\mathbf{L}_o$, this being the domain of *observable labels* of the considered system, obtained by picking up a label from each $\|N\|$, $N \in \mathcal{N}_i$, without violating the ordering imposed by $\mathcal{A}_i$.

Every node $N \in \mathcal{N}_i$ cannot be completely disconnected from nodes in the higher strata (that is, nodes belonging to $\bigcup_{j=1}^{i-1} \mathcal{N}_j$) since N would be unrelated with respect to the nodes in the higher strata, which contradicts the hypothesis, according to which F_i is a stratum and, as such, all the unrelated nodes of the nodes in the i-th stratum are in the i-th stratum themselves. In order not to be unrelated with the nodes in the higher strata, it is necessary that $\forall N \in \mathcal{N}_i, \exists N' \mapsto N \in \mathcal{A}_i$, $N' \notin \mathcal{N}_i$. Eq. (7) prevents the existence of any $N' \mapsto N \in \mathcal{A}_i$ such that N' belongs to lower strata than F_i. This implies that, $\forall N \in \mathcal{N}_i, \exists N' \mapsto N \in \mathcal{A}_i$, $N' \in \bigcup_{j=1}^{i-1} \mathcal{N}_j$. Moreover, each such N' cannot be the parent node of any other node N'', $N'' \in \bigcup_{j=1}^{i-1} \mathcal{N}_j$, since this would imply that N is unrelated with respect to N'', which contradicts the hypothesis, according to which F_i is a stratum and, as such, all the unrelated nodes of N belongs to $\mathcal{N}_i$. Therefore, any N' is bound to be a *leaf* node of a higher stratum, this being a node which is not

the source of any arc in $\bigcup_{j=1}^{i-1} \mathcal{A}_j$. However, there cannot be any leaf node in all the strata F_j, $j \in [1\,..\,i-2]$, since such a leaf node would be unrelated with the nodes in strata F_s, $s \in [j+1\,..\,i-1]$, which contradicts the hypothesis. Therefore, any N' is bound to be a *leaf* node of F_{i-1}.

On the other hand, any leaf node of F_{i-1} is bound to be the parent node of a node in F_i. In fact, a leaf node of F_{i-1} cannot be a final node of the whole observation, since final nodes are bound to belong to the last stratum (if they belonged to another stratum, they would be unrelated with the nodes in the lower strata, which contradicts the stratification hypothesis). Analogously, a leaf node of F_{i-1} cannot be the parent node of any node belonging to strata $F_k, k > i$, since it would be unrelated with respect to all the nodes belonging to the strata F_r, $r \in [i\,..\,k-1]$, which contradicts the stratification hypothesis.

Then, all the leaf nodes of F_{i-1} have a child node at least in F_i, and all the nodes of F_i have a parent node at least in F_{i-1}. Therefore, each signature in $\|\mathcal{O}_{[i]}\|$, is obtained by extending a signature in $\|\mathcal{O}_{[i-1]}\|$, relevant to a path ending in leaf node N' of F_{i-1}, with a signature postfix of F_i, relevant to a path beginning in a node N of F_i that is a child node of N'. This way, all the signature postfixes of F_i are exploited. Since there may be signature postfixes that equal the null sequence, each signature in $\|\mathcal{O}_{[i]}\|$ either is an extension of a signature in $\|\mathcal{O}_{[i-1]}\|$ or equals a signature in $\|\mathcal{O}_{[i-1]}\|$, which proves the thesis. □

Example 10. Fragmentation $\bar{\mathcal{O}}^*$ defined in Example 6 is *not* a stratification as the first stratum $\{N_1, N_3\}$ does not include N_2, the unrelated node of N_3. A possible stratification of $\bar{\mathcal{O}}$ is defined by the following sequence of sets of nodes: $\langle\{N_1\},\{N_2,N_3\},\{N_4\},\{N_5\}\rangle$. As expected by Proposition 3, the signatures of the relevant sub-observations grow monotonically, in fact $\|\bar{\mathcal{O}}_{[0]}\| = \{\epsilon\}$, $\|\bar{\mathcal{O}}_{[1]}\| = \{a\}$, $\|\bar{\mathcal{O}}_{[2]}\| = \{abc, acb\}$, $\|\bar{\mathcal{O}}_{[3]}\| = \{abc, acb, abcd, acbd\}$, and $\|\bar{\mathcal{O}}_{[4]}\| = \{abca, acba, abcda, acbda\}$. □

Proposition 3 states a conceptual property, while it does not imply in any way that the extension of a stratified observation be computed in order to solve a monitoring problem. The incremental growing of candidate signatures entails monotonic monitoring, as stated by Proposition 4.

Proposition 4. *A monitoring problem involving a stratified observation is monotonic.*

Proof. Let $\mu(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}^*, \mathcal{R})$ be a monitoring problem involving the stratified observation $\mathcal{O}^* = \langle F_1, \ldots, F_n\rangle$. According to Eq. (10), the solution is $\Delta(\mu(\Sigma)) = \left\langle \Delta(\wp_{[0]}(\Sigma)), \ldots, \Delta(\wp_{[n]}(\Sigma))\right\rangle$, where $\wp_{[i]}(\Sigma) = (\Sigma_0, \mathcal{V}, \mathcal{O}_{[i]}, \mathcal{R})$, $i \in [0\,..\,n]$. Based on Eq. (5), $\Delta(\wp_{[i]}) = \{\delta_i \mid \delta_i = h_i \otimes \mathcal{R}, h_i \in Bhv(\Sigma, \Sigma_0), h_i \boxtimes \mathcal{V} \in \|\mathcal{O}_{[i]}\|\}$. According to Proposition 3, $\forall i \in [1\,..\,n]$, $\|\mathcal{O}_{[i]}\|$ is composed of all the signatures in $\|\mathcal{O}_{[i-1]}\|$ (possibly) extended with further observable labels. Therefore, each $h_i \in Bhv(\Sigma, \Sigma_0)$, $h_i \boxtimes \mathcal{V} \in \|\mathcal{O}_{[i]}\|$ is a (possible) extension of a history $h_{i-1} \in Bhv(\Sigma, \Sigma_0)$, $h_{i-1} \boxtimes \mathcal{V} \in \|\mathcal{O}_{[i-1]}\|$. Consequently, each diagnosis $\delta_i = h_i \otimes \mathcal{R}$ is a superset of a diagnosis $\delta_{i-1} = h_{i-1} \otimes \mathcal{R}$, where h_i is an extension of h_{i-1}. Since, according to Eq. (5), $\delta_{i-1} \in \Delta(\wp_{[i-1]}(\Sigma))$, this proves that $\forall \delta_i \in \Delta_i, \exists \delta_{i-1} \in \Delta_{i-1}$ $(\delta_{i-1} \subseteq \delta_i)$. As already remarked, since $\Delta_n = \Delta(\wp_{[n]}(\Sigma)) = \Delta(\wp(\Sigma))$, based on Proposition 2, the actual diagnosis $\delta \in \Delta_n$. Therefore, $\exists \delta_{n-1} \in \Delta_{n-1}$ $(\delta_{n-1} \subseteq \delta)$. Analogously, $\exists \delta_{n-2} \in \Delta_{n-2}$ $(\delta_{n-2} \subseteq \delta_{n-1})$, and so on, that is, Condition (11) is fulfilled and, therefore, the proposition is proved. □

Example 11. Consider a variant $\mu'(\bar{\Sigma})$ of the monitoring problem $\mu(\bar{\Sigma})$ defined in Example 7, where $\bar{\mathcal{O}}^*$ of the latter is replaced by the stratified observation introduced in Example 10. Then, $\Delta(\mu'(\bar{\Sigma})) = \langle \Delta'_0, \Delta'_1, \Delta'_2, \Delta'_3, \Delta'_4\rangle$, where $\Delta'_0 = \{\emptyset\}$, $\Delta'_1 = \{\emptyset, \{w\}, \{x\}, \{y\}\}$, $\Delta'_2 = \{\{\mathrm{w}\}, \{x\}, \{x, y\}\}$, $\Delta'_3 = \{\{\mathrm{w}\}, \{x\}, \{x, y\}, \{x, y, z\}\}$, and $\Delta'_4 = \{\{\mathrm{w}\}, \{x, y, z\}\}$. As expected by Proposition 4, $\mu'(\bar{\Sigma})$ is monotonic. □

Given a nontrivial stratified observation $\mathcal{O}^*$, it is always possible to transform it into a different stratified observation by aggregating, for instance, contiguous strata in $\mathcal{O}^*$. In other words, the property of stratification is conserved when several contiguous fragments are grouped together to form coarser-grained fragments. The contrary is not true: when two or more contiguous fragments are obtained by splitting a single stratum, stratification may be lost. In monitoring, we may be interested in considering the *finest stratification*, where strata cannot be further split without losing stratification. This allows the monitoring task to be as reactive as possible in generating candidate diagnoses.

Proposition 5. *The finest stratification is unique.*

Proof. We have already underlined that, owing to Condition (7), a fragmentation is univocally identified by the sequence of $\mathcal{N}_i$ of its fragments. Therefore, a stratification, being a fragmentation whose fragments are called strata, is univocally identified by the sequence of $\mathcal{N}_i$ of its strata.

The finest stratification $\mathcal{O}^*$ of a given observation $\mathcal{O} = (\mathcal{N}, \mathcal{A})$ is the one such that each stratum contains all and only the nodes that are bound to belong to the same stratum. We define the dyadic operator *same-stratum* $\diamond$ between two nodes $N, N' \in \mathcal{N}$ this way: $N \diamond N'$ iff it is necessary that N' belongs to the same stratum as N.

The same-stratum relation is reflexive since by definition, a fragmentation produces a partition of $\mathcal{N}$, where each fragment (a stratum in a stratification) is inherent to a part; therefore, each node $N \in \mathcal{N}$ necessarily belongs to just one stratum; in other words, N needs to belong to the same stratum as N itself.

The same-stratum relation is symmetric. In fact, if $N' \in Unrl(N)$, then $N \diamond N'$. $N' \in Unrl(N)$ means that $N' \notin Anc(N)$ and $N' \notin Desc(N)$, which imply $N \notin Desc(N')$ and $N \notin Anc(N')$, respectively. In other words, $N' \in Unrl(N)$ implies $N \in Unrl(N')$, where the latter entails $N' \diamond N$, thus proving the symmetric property.

The same-stratum relation is transitive. If $N' \in Unrl(N)$, then $N \diamond N'$. Analogously, if $N'' \in Unrl(N')$, then $N' \diamond N''$. If the two previous conditions hold, since Condition (12) has to be fulfilled for every node belonging to the same stratum, it follows that $N \diamond N''$, which proves transitiveness.

Hence, same-stratum partitions $\mathcal{N}$ into equivalence classes, each of which identifies a minimal stratum. □

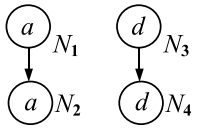


Figure 4: Disconnected observation for $\bar{\Sigma}$.

Example 12. Note how the stratified observation in Example 10 is in fact the finest one. A new stratification can be obtained from it by aggregating the two intermediate strata into a single stratum, thereby leading to the stratification $\langle\{N_1\}, \{N_2, N_3, N_4\}, \{N_5\}\rangle$. □

We may wonder whether all observations can be nontrivially stratified. The answer to this question comes from Proposition 6.

Proposition 6. *Given an observation $\mathcal{O}$ consisting of two nodes at least, a nontrivial stratification of $\mathcal{O}$ exists if and only if $\mathcal{O}$ is connected.*

Proof. The proposition consists of two statements: (i) If $\mathcal{O}$ is connected, then a nontrivial stratification exists; and (ii) If $\mathcal{O}$ is disconnected, its finest stratification is the trivial stratification. Based on the proof of Proposition 5, the minimal strata of an observation $\mathcal{O} = (\mathcal{N}, \mathcal{A})$ are defined by the equivalence classes of nodes identified by the *same-stratum* relation. In case (i), the observation necessarily includes a pair of nodes such as one is the parent of the other. These nodes belong to two distinct equivalence classes identified by same-stratum, therefore a nontrivial stratification exists. In case (ii), the observation consists of several DAGs. Each node N in one of such DAGs needs to belong to the same stratum as whichever node belonging to a distinct DAG, since all the nodes of the other DAGs are unrelated to it. This means that $N \diamond N'$ for whichever pair of nodes N and N' that belong to distinct DAGs of the same disconnected observation. Therefore, all the nodes in $\mathcal{N}$ belong to the same equivalence class, i.e. to the same stratum. □

Example 13. Let $\mu''(\bar{\Sigma}) = (\Sigma_0, \bar{\mathcal{V}}, \tilde{\mathcal{O}}^*, \bar{\mathcal{R}})$ be a monitoring problem, where $\tilde{\mathcal{O}}^* = \langle\{N_1, N_2\}, \{N_3, N_4\}\rangle$ is a fragmentation of the observation displayed in Fig. 4. It can be easily verified that $\Delta(\mu''(\bar{\Sigma})) = \langle\Delta_0'', \Delta_1'', \Delta_2''\rangle$, where $\Delta_0'' = \{\emptyset\}$, $\Delta_1'' = \{\{w\}, \{x\}, \{y\}\}$, $\Delta_2'' = \{\{w, z\}\}$, therefore the monitoring problem is monotonic although the observation is not stratified. This, however, does not conflict with Proposition 6 since, by definition, a stratified observation guarantees monotonicity *independently of the specific system at hand*. In other words, a stratified observation is a sufficient condition for monotonicity, not a necessary one. □

A final question is about the nature of the solution of a monitoring problem involving a stratified observation. As we know, $\Delta(\mu(\Sigma))$ is a sequence $\langle\Delta_0, \Delta_1, \ldots, \Delta_n\rangle$ of sets of candidate diagnoses, with each Δ_i being the solution of the diagnostic sub-problem $\wp_{[i]}(\Sigma)$. So, *What is the relation between Δ_i and Δ_{i+1}?* Proposition 7 gives the answer.

Proposition 7. *Let Δ_i and Δ_{i+1}, $i \in [0\,..\,(n-1)]$, be two consecutive elements in the solution of a stratified monitoring problem. Then,*

$$\forall \delta' \in \Delta_{i+1}\ (\delta' \supseteq \delta, \delta \in \Delta_i). \tag{13}$$

In other words, the set of candidate diagnoses in Δ_{i+1} is obtained by extending a subset of the candidate diagnoses in Δ_i. This is a *shrink and expand* operation, where Δ_i is first shrunk and then the remaining candidates are possibly extended with additional faults, to eventually make up Δ_{i+1}.

The proof of Proposition 7 is included in that of Proposition 4 since the former proposition is a Corollary of the latter. Actually, Condition (13) is a definition of monotonic monitoring alternative with respect to Condition (11). In fact, Condition (11) constrains the growth of the actual diagnosis to be monotonic. However, the actual diagnosis is unknown, and variable n in (11) could be assigned any non negative integer value (that is, we do not know when the observation is complete). Therefore, Condition (11) has to hold $\forall \delta = \delta', \delta' \in \Delta_{i+1}, i \geq 0$, thus obtaining Condition (13).

Let Γ_i be the graph including all and only the histories entailing Δ_i. We can dually say that $\mu(\Sigma)$ is monotonic iff $\forall i \in [0\,..\,(n-1)]$, $\forall h_i \in \Gamma_i$, $\exists h_{i-1} \in \Gamma_{i-1}$ such that h_{i-1} is a prefix of h_i. In summary, monotonicity of candidate diagnoses is a consequence of the monotonic growing of candidate histories, which, in turn, is a consequence of the monotonic growing of the observation extensions.

Conclusion

This paper deals with monitoring-based diagnosis of DESs under uncertain temporal observations, however, it neither proposes any computational method in order to carry out the task nor is focused on the notion of an uncertain observation. Its perspective is rather a conceptual one, so as to span on several approaches to the task. Its main points can be summarized as follows:

- The monotonicity property of the solutions (histories and/or fault sets) of a monitoring-based diagnosis problem has been defined, which is important from the point of view of the dependability of such results.
- While multiplicity of diagnosis results depends on the non-determinism and partial observability of the system as well as on the (logical, temporal, and source) uncertainty of the observation, it has been shown that, the lack of monotonicity depends on the incompleteness and temporal uncertainty of the observation.
- A sufficient condition has been explicated, under which incompleteness and temporal uncertainty of the observation can be coped with by any sound and complete monitoring-based diagnosis method in order to achieve monotonicity of system-level candidate diagnoses, thus leading to the notion of a stratified observation. Summing up, results are surely monotonic if two conditions hold: the problem-solving method is sound and complete, and the temporal observation is stratified.

By system-level candidate diagnose we mean either a history of the whole system or the set of faults derivable by such a history. A system-level candidate diagnose produced by a sound and complete method is consistent with the considered observation as well as with the considered system, in other words, it is globally consistent (Su & Wonham 2005).

Monotonicity can be regarded as a comparison criterion for different approaches to monitoring-based diagnosis of DESs, when they take into account temporally uncertain observations, such as (Lamperti & Zanella 2005; Pencolé & Cordier 2005). The former adopts a sound and complete method, but processes an (uncertain) observable event (and not an observation stratum) at each monitoring step. Therefore, such an approach is not monotonic. Its adaptation so as to take as input stratified observations is an engagement for the future.

The latter work, after (Pencolé, Cordier, & Rozé 2001), processes a packet of observable events instead of just one event at each monitoring step. In (Pencolé & Cordier 2005) the system observation taken into account at each step is that received within the time span of a so-called temporal window. It consists of a sequence of logically-certain observable events for each component, plus a set of temporal constraints that can lead to envisage a partial order of the observable events generated by each considered (sub)system. The unsolved problem is how to frame temporal windows in order to achieve completeness of diagnostic results. This means that, given the modeling primitives and the reasoning mechanism of the approach, completeness depends on how the observation is fragmented over time. As far as this problem is not solved, the method cannot achieve monotonicity in the general case, as it cannot guarantee completeness beforehand and, then, the actual diagnosis may be missed.

The content of the present work is related with that of (Grastien, Cordier, & Largouët 2005), which deals with the slicing of an uncertain observation represented as an automaton. Such a slicing is correct if it guarantees the completeness of the set of candidate histories when the task of a-posteriori diagnosis is performed by either a modular or an incremental method. If the incremental approach is applied also on-line, as suggested by the authors, results are monotonic (although the paper never discusses monotonicity).

A final remark as to the concept of a stratified observation is worthwhile. If the observation taken as input by a sound and complete a-posteriori diagnosis method is stratified, several monotonic intermediate results can be produced. If an intermediate singleton is produced (a unique candidate diagnosis), this means that the faults it includes are certain (in other words, such a diagnosis is a subset of the actual one). More generally, the intersection of the candidate diagnoses produced at any intermediate time is included in the actual diagnosis. This is quite interesting for making domain-dependent reasoning and/or taking decisions (such as, reconfiguration and control) before the diagnostic process is over. Therefore, another intent for the future is to investigate the relationships between diagnosability of DESs and monotonicity of results.

References

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

Console, L.; Picardi, C.; and Ribaudo, M. 2002. Process algebras for systems diagnosis. *Artificial Intelligence* 142(1):19–51.

Debouk, R.; Lafortune, S.; and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamic Systems: Theory and Application* 10:33–86.

Grastien, A.; Cordier, M.; and Largouët, C. 2005. Incremental diagnosis of discrete-event systems. In *Sixteenth International Workshop on Principles of Diagnosis – DX'05*, 119–124.

Lamperti, G., and Zanella, M. 2000. Diagnosis of discrete-event systems with uncertain observations. In *Fourth IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes – SAFEPROCESS'2000*, 1109–1114.

Lamperti, G., and Zanella, M. 2002. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence* 137(1–2):91–163.

Lamperti, G., and Zanella, M. 2004. A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* 34(5):2222–2244.

Lamperti, G., and Zanella, M. 2005. Monitoring and diagnosis of discrete-event systems with uncertain symptoms. In *Sixteenth International Workshop on Principles of Diagnosis – DX'05*, 145–105.

Lamperti, G., and Zanella, M. 2006. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence* 170(3):232–297.

Lunze, J. 2000. Diagnosis of quantized systems based on a timed discrete-event model. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans* 30(3):322–335.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence* 164:121–170.

Pencolé, Y.; Cordier, M.; and Rozé, L. 2001. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *Twelfth International Workshop on Principles of Diagnosis – DX'01*, 151–158.

Rozé, L., and Cordier, M. 2002. Diagnosing discrete-event systems: extending the 'diagnoser approach' to deal with telecommunication networks. *Journal of Discrete Event Dynamic Systems: Theory and Application* 12:43–81.

Rozé, L. 1997. Supervision of telecommunication network: a diagnoser approach. In *Eighth International Workshop on Principles of Diagnosis – DX'97*, 103–111.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology* 4(2):105–124.

Sampath, M.; Lafortune, S.; and Teneketzis, D. 1998. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control* 43(7):908–929.

Su, R., and Wonham, W. 2005. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control* 50(12):1923–1935.

Models and Tradeoffs in Model-Based Debugging

Wolfgang Mayer and **Markus Stumptner**
University of South Australia
Advanced Computing Research Centre
Mawson Lakes, SA 5095, Australia
mayer|mst@cs.unisa.edu.au
Fax: (++61) (08) 8302 3988

Abstract

Developing model-based automatic debugging strategies has been an active research area for several years that aims at locating defects in a program, utilising the fully automated generation of an declarative model of the program semantics from source code. We provide an overall comparison of past developments in model-based debugging and predicate abstraction based frameworks and assess strengths and weaknesses of the individual approaches. An empirical comparison is presented that investigates the relative accuracy of different models on a set of test programs and fault assumptions, showing that our abstract interpretation based model may provide a trade-off between accuracy and computational complexity. Further, compare selected model-based debugging techniques with other state-of-the-art automated debugging approaches and outline possible future developments in automatic debugging using model-based reasoning frameworks.

Introduction

Developing tools to support the software engineer in locating bugs in programs has been an active research area during the last decades, as increasingly complex programs require more and more effort to understand and maintain them. Several different approaches have been developed, using syntactic and semantic properties of programs and languages. While a number of different approaches (slicing (Tip 1995), mutation testing (Offutt *et al.* 1996), model-checking (Chaki, Groce, & Strichman 2004), Delta-debugging (Cleve & Zeller 2005), trace-based analyses (He & Gupta 2004) and model-based debugging (Friedrich, Stumptner, & Wotawa 1996)) have been introduced, each addresses a particular deficiency of previous approaches and no systematic comparison of the strengths, weaknesses and relationships between the individual debugging methods has been undertaken.

In this paper, we begin the task by summarising different approaches to *model-based debugging* that have been proposed in the literature. Focusing on imperative and object-oriented languages, we briefly outline the main modelling paradigms and explore relationships between different models with respect to empirical results.

Model-based Debugging

Model-based software debugging (MBSD) is an application of *Model-based Diagnosis (MBD)* (Reiter 1987) techniques to locating errors in computer programs. MBSD was first introduced by Console et. al. (Console, Friedrich, & Dupré 1993), with the goal of identifying incorrect clauses in logic programs; the approach has since been extended to different programming languages, including VHDL (Friedrich, Stumptner, & Wotawa 1996) and Java (Wieland 2001). The key aspects of the approach are that –unlike verification techniques– it does not intrinsically require a separate formal specification, but is intended to operate solely on the outcome of test cases (test vectors); a formal representation of the program (a "model") is built automatically, and used not to determine whether errors exist, but to try and locate them in the actual code.

The basic principle of MBD is to compare the *model*, a description of the correct behaviour of a system, to an *observed behaviour* of the system. Traditional MBD systems receive a description of the observed behaviour through direct measurements while the model is supplied by the system's designer. The difference between the behaviour anticipated by the model and the actual observed behaviour is used to identify components that, when assumed to deviate from their normal behaviour, may explain the observed behaviour.

The key idea of adapting MBD for debugging is to exchange the roles of model and actual system: the model reflects the behaviour of the (incorrect) program, while test cases specify the correct result. Differences between values computed by the program and values anticipated by test cases are used to compute model elements that, when assumed to behave differently, explain an observed misbehaviour. The program's instructions are partitioned into a set of *model components* which form the building blocks of explanations. Each component corresponds to a fragment of the program's source text, and diagnoses can be expressed in terms of the original program to indicate a potential fault to the programmer (see Figure 1).

Each component can operate in *normal mode*, where the component functions as specified in the program, or in one or more *abnormal modes*, with different behaviour. Intuitively, each component mode corre-

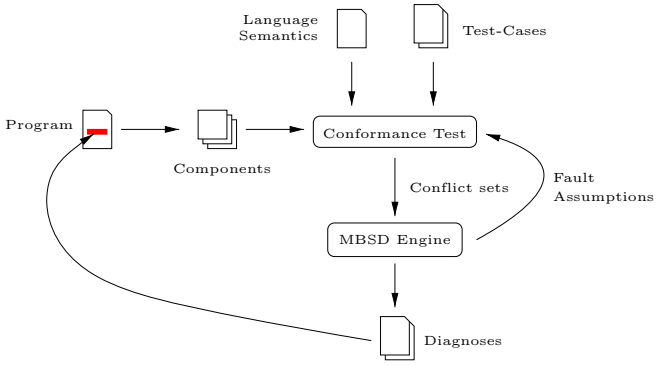


Figure 1: MBSD cycle

sponds to a particular program modification.[1]

The model components, a formal description of the semantics of the programming language and a set of test cases are submitted to the conformance testing module to determine if the program reflecting the fault assumptions is consistent with the test cases. A program is found *consistent* with a test case specification if the program *possibly* satisfies behaviour specified by the test case.

In case the program does not compute the anticipated result, the MBSD engine computes possible explanations in terms of mode assignments to components and invokes the conformance testing module to determine if an explanation is indeed valid. This process iterates until one (or all) possible explanations have been found.

Formally, our MBSD framework is based on Reiter's theory of diagnosis (Reiter 1987), extended to handle multiple fault modes for a component. MBSD relies on test case specifications to determine if a set of fault assumptions is a valid explanation, where each test case describes the anticipated result for an execution using specific input values.

While the overall process is intuitive, the precise test if a program variant conforms to a test specification is undecidable in general and must be approximated. Different abstractions have been proposed in the literature, ranging from purely dependency-based representations (Friedrich, Stumptner, & Wotawa 1996) to Predicate Abstraction (Köb, Chen, & Wotawa 2005).

In the following, the major modelling approaches are summarised briefly. Readers interested in a more detailed presentation are referred to (Mayer & Stumptner 2007).

Dependency-based Models

Models derived from dependencies between program statements were among the first to be developed. Dependency models are constructed from dependencies between statements in a program, either by means of static analysis or through analysis of particular program executions (Wieland 2001).

An abstract model of the program is built that models the flow of correct and incorrect values through a program. Concrete values computed by a program are abstracted into a lattice where only *correct* and *incorrect* can be distinguished. Parametrised with Boolean variables representing fault assumptions introduced by the diagnostic engine and transformed to a representation in propositional logic, the model is used to ascertain whether the program variant reflecting particular fault assumptions is possibly consistent with the test specification.

While the dependency-based models can be applied efficiently even to large programs ((Friedrich, Stumptner, & Wotawa 1996) successfully applied dependency-based models to VHDL-RTL programs of up to several MB of source code), object-oriented programs often result in many spurious explanations. This can be attributed to the coarse abstraction of the concrete semantics into abstract transitions computing only *correct* and *incorrect*. In particular for long chains of interdependent program fragments with no assertions or other observations in-between, the abstraction is too weak to determine that a particular candidate is inconsistent with the test specification and consistency must be assumed. Other purely dependency-based debugging techniques, such as Slicing, frequently lead to large result sets due to the same drawbacks.

Value-based Models

The value-based approach (Mateis, Stumptner, & Wotawa 1999) attempts to eliminate spurious explanations by strengthening the conformance test. The simple lattice used for dependency-based models is replaced with a version borrowed from constant propagation, where values that are precisely known are represented accurately; no information is derived in case values are not uniquely determined given the fault assumptions and test specification.

Comparing values computed by a program reflecting fault assumptions with the values expected by a test specification, explanations that cannot fully account for all failed test cases can be eliminated. It can be shown (Mayer & Stumptner 2007) that this approach is strictly more precise than the dependency-based formalism, leading to fewer spurious explanations.

While value-based conformance checking is more precise than dependency-based approximation, the models are computationally more expensive. The time required to ascertain consistency in a dependency-based model is proportional to the program size; the complexity of the value-based representation is a function of the length of the execution trace, as in the worst case, the complete test execution must be simulated. Consequently, value-based models are only applicable for small programs (up to a few hundred components).

Concrete values allow to effectively diagnose programs where the control and data flow can be determined statically. For programs making heavy use of

[1] The main difference to Mutation Testing (Offutt *et al.* 1996) is that MBSD typically does not assume a specific behaviour for the replacement program expression.

dynamically created data structures, loops and recursive methods, the approach is less effective, as a single fault assumption can render large parts of a program's control flow and program state unknown, collapsing the model and prohibiting the detection of inconsistencies.

Abstraction-based Models

To overcome limitations inherent to previous approaches, the conformance tester can be amended to predict values of a program's state at an abstract level in case the precise value cannot be ascertained. Two different frameworks have been proposed: (i) an extension to the value-based model using the Abstract Interpretation (Cousot & Cousot 1977) framework, and (ii) a Predicate Abstraction based analysis (Köb, Chen, & Wotawa 2005).

The idea behind the abstract interpretation based model (AIM) (Mayer & Stumptner 2003) is to combine static analysis and test execution to approximate possible execution traces through a program. Instead of a unique execution trace, each combination of test case and fault assumption may induce graph representing a set of consistent traces. Static analysis techniques are applied to those sections of an execution trace where precise values cannot be determined. Since fault assumptions in many cases affect only a small part of a program's state, the analysis is confined to small regions of a trace. Precise values derived in other parts of the execution trace can be exploited to replace complex relational abstractions with simpler non-relational approaches without suffering from degrading results. While traditional abstract interpretation is prohibitively expensive for many programs, combining concrete and abstract execution helps to keep the number of possible traces small. Results obtained by abstract interpretation (presented in the *Empirical Evaluation* section) demonstrate that results can be improved slightly compared to the previous approaches.

The abstract interpretation approach has limited value for abstract execution traces where multiple related values are affected by a single fault assumption. In particular, fault assumptions affecting the dynamic allocation of data structures may have vast impact on the program execution. To eliminate such spurious explanations, additional information that cannot be derived from the program is required.

As the AIM may abstract away information that is vital to isolate faults in some programs, a different approach that incrementally refines the abstraction to suit the program has been proposed (Köb, Chen, & Wotawa 2005). An additional analysis based on Predicate Abstraction and counter-example guided abstraction refinement (Clarke *et al.* 2003) is performed in case the value-based model cannot derive a conflict. We refer to the resulting model as *PAM*. Replacing the abstract interpretation engine with an iterative predicate abstraction process, model checking is able to provide improved results for selected programs (Köb, Chen, & Wotawa 2005). While the initial results are encouraging, it is not clear whether the predicate refinement approach is effective in the presence of partially specified programs due to fault assumptions. Further, the impact of fault assumptions on the abstraction refinement process has not been explored thoroughly.

Model Checking

A different approach using bounded model checking (BMC) has been presented in (Griesmayer, Staber, & Bloem 2006). While not presented in terms of model-based debugging and diagnosis, the approach is in principle quite similar. The architecture of the system differs slightly from the one presented here, where the debugging engine and the conformance test are combined into a single model checking procedure based on propositional satisfiability. The program is transformed such that all execution paths up to a pre-set length can be represented without method calls and loops. The resulting program is encoded as a logic formula, where conditionals and free variables introduced in the compilation process model fault assumptions and their effects on program states. Values for these variables are not implied by the program behaviour, but are controlled by the model checker. An incremental search is performed to find all consistent assignments to variables representing fault assumptions up to a given number of assumptions. Each set of variable assignments corresponds to a minimal diagnosis.

As a side-effect, this method is able to provide suggestions on how to modify the faulty program such that all test specifications are satisfied. While the repair approach is not complete in general, (Griesmayer, Staber, & Bloem 2006) suggests that the results provide helpful insights in many cases.

Comparing the BMC-based approach to PAM, it can be seen that the PAM may result in spurious explanations due to the abstraction used in the modelling process, while BMC does not share this undesirable property, provided execution traces are explored in sufficient depth to cover the relevant program behaviour. If only a single failing test case is available and an explanation e returned by the BMC approach occurs only once in the trace, e is guaranteed to be non-spurious. BMC delivers results that are identical to the set of explanations obtained from the most precise consistency-based model (which cannot be constructed in general) if all explanations satisfy the conditions outlined here. Unfortunately, these conditions need not be satisfied for programs, and it may be difficult to determine a suitable depth limit. Therefore, further investigations in suitable approximations and trade-offs between precision and model complexity are desired.

Fault Assumptions

Most MBSD approaches have great difficulty with faults that manifest as structural differences in the model, such as assignments to incorrect variables or missing program fragments.

While tailored fault modes provide the means to isolate such faults, complex fault modes must be applied

in a controlled manner to avoid exponentially increasing the search space. To reliably and efficiently locate such faults, additional information is required. While formal specifications of certain aspects of the behaviour of a *correct* program, such as automata or modal logic formulae, are most beneficial, these are also difficult to obtain in practice. In particular, in an ad-hoc debugging process, where relevant properties are not known beforehand or may change frequently, relying on (casual) debugger users to be able to provide any kind of formal description is not realistic.

We propose to exploit certain invariants that in isolation provide little information about a program's behaviour, but may allow to exonerate a number of otherwise difficult to eliminate program fragments. Instead of creating strong formal descriptions, we rely on (minimal) pre-existing documentation and selected invariants associated with natural language templates to allow users not trained in formal methods to use our automated debugger.

In particular, simple invariants derived from existing design documentation can be utilised to guide the debugging process (Mayer & Stumptner 2004). For example, type and cardinality constraints, sequence diagrams and pre- or post conditions can be compiled into abstract models that allow to check programs states for consistency, derive conflicts and guide the search for structural faults. An interactive refinement process using simple observations ("... this data structure should not have changed!") can further guide the debugger. The details of this proposal are beyond the scope of this paper; a more detailed discussion of the model and examples are presented in (Mayer & Stumptner 2004).

A comparison outlining the worst-case accuracy of the model-based representation discussed here and their relationship to other models is presented in (Mayer & Stumptner 2007).

Experimental Results

In this section, empirical results are presented to demonstrate the performance of the individual models relative to each other. While we aim at comparing a wide variety of models, the discussion in the remaining paper has been limited to models where either an implementation is available, or results have been published using a test set that can be run on different systems.

In the following, we present empirical experiments on a test set of programs, showing that the BMC-based approach provides the fewest false positives and is closely followed by the AIM. Dependency-based models generally perform poorly on our test set. Subsequently, we illustrate that while the BMC model is very effective for functional faults, that is faults that do not imply assignments to incorrect variables, the AIM is shown to provide more general fault models that are effective for isolating structural faults.

Quality measure. To compare different debugging techniques, an unbiased measure of the quality of explanations is desired. The *Score* indicator has been proposed by (Renieris & Reiss 2003) as an objective measure to compare the performance of different debugging approaches. Intuitively, it is defined as the fraction of a program that need *not* be investigated given the program fragments, R, implicated by a debugging tool such that the true cause, C, of a failure is guaranteed to be found (Renieris & Reiss 2003). The size of a program P is measured as the number of nodes in its *Program Dependence Graph* $pdg(P)$ that represents control and data dependencies between statements and methods. Starting at the implicated statements R, the program is traversed along the data and control dependencies in $pdg(P)$ in breadth-first order until an expression in C is reached:

$$score(P, R, C) = 1 - \frac{|BFS(pdg(P), R, C)|}{|pdg(P)|}.$$

$BFS(pdg(P), R, C)$ denotes the set of nodes in $pdg(P)$ that is traversed.

Similar to simpler indicators, for example the fraction of a program implicated by potential explanations, the score favours concise reports focusing on incorrect program elements, while large reports or reports not indicating a faulty program element are assigned a low score. In particular, if a report implicates only the nodes corresponding to the true cause, then $score(P, R, C)$ is close to one. Conversely, if the report encompasses the entire program or PDG, the score is zero.

Experimental Setup. A set of test programs has been used to evaluate the performance of different debugging approaches. The *TCAS* program was taken from the *Siemens Test Suite*[2], a test bench commonly used in the debugging community, and transcribed to Java. The remaining programs have been retained from earlier tests of the VBM and dependency-based models.

Table 1 summarises the experimental setup. For each program, n variants with different faults were created by seeding single and double faults and programs were tested under different test specifications. All programs were analysed using between one and *Test* test cases. With the exception of *Adder* and *TCAS* (the former simulates a binary full adder circuit and the latter the resolution advisory component of a collision avoidance system used in commercial aircraft), no significant difference between the single and the multiple test-case scenario was detected. This can be attributed to the structure of the selected programs, where a large fraction of statements are executed for all test cases.

A summary of our results is given in Table 1. (The values are 0.5 quartiles over all program variants and available test cases, as well as the total average values.) *LoC* denotes the number of non-comment, non-delimiter lines in the program's source code, *Tests* denotes the number of test cases available to compute

[2] `http://www-static.cc.gatech.edu/...`
`.../aristotle/Tools/subjects/`

explanations, *Comp* represents the number of diagnosis components used to construct explanations, *SSlice*, *DSlice* and *VBM* and *AIM* denote the scores obtained from single-fault explanations computed by the static and dynamic dependency models, the value-based model and the abstract interpretation-based approach, respectively. *Exec* contains the score obtained from the number of statements that are executed for a test run. Due to limitations of the implementation, some valid diagnoses listed for the AIM may not be included in the VBM's results, yielding a slight bias towards the VBM. The columns labelled *Time* denote the time in seconds required to compute explanations using the VBM and the AIM, respectively.

Static vs. Dynamic Dependency-Based Models. The results in Table 1 support the conclusion that that static slicing and dynamic slicing in many cases cannot improve much compared to the entire program and the statements executed in test runs. Similarly, dynamic slices often improve little compared to the executed statements, as all statements contribute to the final result. Given the close relationship between dependency-based models and slicing (Wotawa 2002), this result translates directly to dependency-based MBSD.

Dicing is not applicable for most programs, as correct test executions were not available. (*TCAS* is an exception and is accompanied with an extensive test suite; however, Dicing fails to implicate any program statement, as all incorrect program fragments are contained in both passing and failing test executions.)

The time required to compute explanations is well below 1 s for all dependency-based models and has been omitted from Table 1.

VBM vs. AIM. Comparing VBM and AIM, it can be seen that the AIM improves over the VBM in most cases. In fact, cases where the VBM provides fewer explanations is due to explanations missed by the VBM.[3] Overall, the scores for the AIM consistently outperform the VBM's, indicating that the improved approximation of the conformance test indeed helps to eliminate spurious explanations. The AIM provides significantly fewer explanations than slicing, but is computationally more demanding. The total time required to compute explanations using the AIM ranges from one second to one hour.

AIM vs. BMC. Experiments presented in (Griesmayer, Staber, & Bloem 2006) indicate that the model-checking approach derives slightly fewer explanations than the MBSD framework, but at the cost of scalability. The average number of potential faults for the 41 TCAS programs is 8 for the SAT checking approach, while the MBSD model returns 10 explanations on average. Translated to a score, the VBM results in 0.950, while BMC achieves scores exceeding 0.995 for half of the test cases. This can be explained by the fact that the model-checking approach simulates each path precisely, while the abstraction-based MBSD models rely on approximation. A more precise conformance test implies fewer explanations, but demands more resources.

For programs manipulating dynamic data structures, model checking is able to derive superior results due to the case distinctions performed by the model checker, but may exceed available resources even for simple programs. Suitable definitions of fault modes for expressions computing object references need to be devised to avoid implicating a large number of expressions. (Griesmayer, Staber, & Bloem 2006) report that their model checker failed to deliver results on a program implementing a red-black tree data structure with six nodes due to resource exhaustion.

While most of the BMC model's scores are in $[0.9, 1)$, our results are slightly lower (Table 1). This can partly be explained by the weaker abstraction we apply, but also by different model granularity. Comparing the diagnoses with the number of assumables instead of the size of the program, the two approaches lead to similar distributions. Comparing the execution times, it can be seen that the AIM is faster than BMC (1093 s on average) on the same processor.

Difference-based Debugging. Table 2 compares the AIM to a spectra-based approach to fault localisation proposed in (Renieris & Reiss 2003), where failing test executions are compared to "similar" correct executions to isolate potential faults. Columns two and three represent the distribution of scores computed for the AIM, whereas columns four and five represent scores derived using the spectra-based analysis (in particular, the nearest-neighbour-based analysis, which performed best in the cited study). For each TCAS program, a score was computed based on all available test cases. While a unique score for the model-based approach can be obtained by combining all available test cases in the analysis, the spectra-based approach derives a different score for each revealing test run. Different from the study presented in (Renieris & Reiss 2003), column *NN* contains the *best* possible result that can be achieved using the nearest neighbour analysis rather than the distribution of average scores.

Model-based debugging achieves vastly better results than the spectra-based approach: where nearest neighbour based debugging cannot provide useful results for more than half of the considered programs, model-based fault localisation consistently achieves scores above 0.7. Although the MBSD technique delivers more consistent results, the maximum scores within the same category of the spectra-based approach are slightly higher. Therefore, if a large number of test cases is available that makes it easy to find a correct execution similar to a given faulty trace, difference-based

[3] The implementation of the VBM makes quite strong assumptions on where in a program faults are located; many faults in our test set violate those assumptions and cannot be detected adequately.

Table 1: Experimental results

Name	n	LoC	Tests	Comp	SSlice	DSlice	Exec	VBM	Time (s)	AIM	Time (s)
Adder	5	49	8	31	0.469	0.551	0.387	0.836	6	0.878	7
Binomial	7	80	2	45	0.612	0.612	0.537	0.706	20	0.863	55.5
BinSearch	7	29	4	24.9	0.137	0.275	0.275	0.310	13	0.931	5
BubbleSort	5	29	4	11	0.620	0.620	0.620	0.672	7	0.897	4
Hamming	7	48	3	38	0.208	0.354	0.354	0.666	556	0.875	64
Permutation	5	54	2	25	0.537	0.537	0.537	0.518	10	0.833	25
Polynom	9	105	2	68.9	0.786	0.786	0.650	0.898	1035	0.910	359
SumPowers	6	27	4	16	0.407	0.481	0.481	0.592	7	0.667	2
TCAS	41	78	131	42	0.000	0.853	0.843	n/a	n/a	0.950	133
Average	10.2	55.44	17.78	33.53	0.420	0.563	0.520	0.650	206.8	0.867	72.72

approaches may provide equally good or better results than the model-based approach. Otherwise, model-based reasoning may provide better accuracy.

Cause Transitions. Another approach based on differences between passing and failing test cases is presented in (Cleve & Zeller 2005). The idea pursued in this work is to find "failure-inducing differences" between program states in a correct and a failing test. Relevant differences between states in different execution traces can be seen as vertexes in a causal chain that may explain why one program execution failed whereas another execution succeeded. Bisection search techniques can be applied to refine causal chains and implicate additional expressions in a program. Faults are located by traversing dependencies between program statements starting at the last isolated cause transition. A comparison of cause transition based debugging and spectra-based fault localisation is presented in (Cleve & Zeller 2005), identifying the cause transitions method as superior. Direct comparison with model-based techniques would be desirable; however, individual results for cause transitions applied to TCAS have not been published.

Results aggregated over the entire Siemens suite show that the distribution of scores obtained using model-based debugging is similar to the cause-transition-based approach *for the interval [0.9,1)* (the best cause transition method achieves scores for roughly 30% of all test programs). In approximately five percent of the test cases, cause transitions are able to pinpoint the location of a fault precisely – a result consistency-based approaches and spectra-based debugging cannot deliver. The remaining results obtained from cause transitions are mainly distributed in the interval $[0, 0.6)$; scores within $[0.6, 0.9)$ are distributed evenly and can be expected in roughly 20% of all cases. The comparison based on the distribution of results given here may not be fully accurate, as the wider scope of the analysis in (Cleve & Zeller 2005) may lead to different results than TCAS only.

Overall, it seems that cause transitions are highly accurate for faults where program states close to the true fault can be inspected for both the correct and the failing program execution; otherwise, cause transitions seem to result in relatively low scoring explanations.

Table 2: AIM vs. spectra-based fault localisation

Score	AIM	%	NN	%
[0,0.1)	0	0	19	46
[0.1,0.2)	0	0	0	0
[0.2,0.3)	0	0	0	0
[0.3,0.4)	0	0	0	0
[0.4,0.5)	0	0	3	7
[0.5,0.6)	0	0	0	0
[0.6,0.7)	0	0	4	10
[0.7,0.8)	12	29	1	3
[0.8,0.9)	15	37	5	12
[0.9,1]	14	34	9	22

Table 3: Results of fault isolation by distance metrics

TCAS	explain	Δ-slicing
v1	0.51	0.91
v11	0.36	0.93
v31	0.76	0.93
v40	0.75	–
v41	0.68	0.88

Distance Metrics. Another difference-based approach is presented in (Groce 2004), where abstract traces derived by a model-checker are compared to isolate potential faults. The approach extends (Groce & Visser 2003) such that execution traces similar to a known counter example are explored with the goal of isolating a minimal difference that induces a transition from a correct to a faulty execution trace. Column *explain* in Table 3 summarises results for five TCAS programs obtained using the *explain* approach (Groce & Visser 2003) and column Δ-*Slice* presents results obtained from the difference-based algorithm (Groce 2005).

The AIM outperforms the *explain* approach on all five examples with equivalent results on version 31. Opposite results are obtained for the distance-based debugger, where the AIM consistently falls short by up to 16%. This is most likely a result of a more precise representation of the semantics of the underlying program, together with the requirement of an additional specification of the correct behaviour that allows to search for correct executions that are close to a given counter example.

Finding Multiple Faults. Initial experiments with multiple seeded faults indicated that the number of explanations obtained from the VBM and AIM does not differ significantly from static slicing and does not warrant the additional overhead necessary for the complex models. Therefore, developing techniques that can deal effectively with multiple defects over a wide range of programs remains for future investigation.

Structural Faults

While the test suite described in the previous paragraphs occasionally contains assignment faults, missing or additional statements, a systematic evaluation of the models' ability to locate structural faults is desired. For this purpose, a test bench consisting of 38 variants of the TCAS program were created, each containing a single assignment fault where the target variable of an assignment was changed randomly.

The AIM was applied to each of the variants, using ten revealing test cases to drive the diagnostic process. In addition to the plain AIM applied in previous sections, we extended the model with additional fault models tailored to locate faults in assignment statements affecting variables and data structures reachable from local and globally visible variables. Specifically, we introduced the following fault modes:

RHS. Assume the right-hand side expression of an assignment is incorrect. The value stored by an assignment is left unspecified, while the target variable remains unchanged. This is the basic fault model used in the evaluation described previously.

LHS. Each assignment statement assumed to exhibit the *LHS* fault may assign to any type-compatible local or global variable accessible at the assignment's label in the program. The right-hand-side expression of the assignment is assumed to be correct.

GSF (Global Structural Faults). In addition to the effects of mode *LHS*, the right hand side expressions of each assignment in mode *GSF* is assumed incorrect.

To simulate complex structural faults in a method, each method call in mode *GSF* may assign an arbitrary value to a variable visible at the call site. This fault model allows to isolate a method as possible source of a structural fault, without modelling the method on a detailed level.

Table 4 presents a summary of the results obtained under different fault assumptions. For each fault model, the median number of diagnoses per test case, the total percentage of test cases where the fault was among the implicated program elements, the average score and time in seconds for a test case are presented.

Using the *RHS* mode only, 8 out of 38 faults could be located; the results obtained with the remaining models included all seeded faults. While *RHS* returns fewer explanations, the overall score of the fault mode is lower, since seeded faults are not always correctly identified. On average, *RHS* and *LHS* derived between 5 and 6 explanations for each test case, while the *GSF* mode implicates roughly three times the number of statements.

Table 4: Result for Assignment Faults

Model	Diagnoses	Located	Score	Time (s)
RHS	5	21 %	0.840	12
LHS	6	100 %	0.923	44
GSF	18	100 %	0.769	51

The time difference between *GSF* and the other modes can be attributed to the particular implementation of the model, which does not cache the set of accessible variables.

While *RHS* and *LHS* result a reasonable number of explanations, *GSF* clearly is too general to be applied widely. However, the fault model may be useful when restricted to certain complex program elements, such as method calls. Then, complex faults, such as multiple faults or missing code, within the contained program fragment may be isolated, which could not be achieved otherwise.

First experiments indicate that the BMC-based model is too precise to effectively deal with structural faults: we were unable to solve a single instance of the same debugging problem using the *LHS* fault mode. Given that the model checker has to explicitly examine all possible choices of target variables for each mode assignment, the state space becomes prohibitively large. None of the resulting SAT instances could be solved due to memory exhaustion.

The results indicate that specialised models are required to effectively locate faults inducing changes in a program's data flow. It is also apparent that unless the scope of a potential structural fault can be restricted to a narrow region of a program, precise models are of limited use. Instead, diagnoses computed using more coarse grained models on a higher level of abstraction can be used in a preliminary step to focus the modelling efforts. Furthermore, the abstraction level must be chosen carefully to ensure the search space remains small while not losing too much accuracy. This is important in particular for programs manipulating dynamic data structures, as then the set of target variables increases rapidly and explicit testing becomes infeasible.

Conclusion

We introduced the basic principle of model-based software debugging and illustrated different models centred around abstract simulation of programs. Differences to previous approaches were outlined and results obtained using different debugging strategies were compared. We illustrated the relationship between dependency-based approaches that can be applied to large programs but lead to coarse results to execution based and abstract frameworks, which lead to more refined approaches but are computationally more expensive.

Further research is needed to find a model representation that lies between the execution-based modelling paradigm and the predicate abstraction based formalisms to obtain a debugging framework that can effectively handle a variety of programs. Synergies

between test execution and static analysis can be exploited to limit the scope of the analysis to relevant fragments of the execution trace, avoiding the state space explosion encountered for model-checking while avoiding overly coarse approximation that can be encountered for static analysis based frameworks.

While MBSD has been shown to focus user attention to a small number of fault candidates, no single approach is likely to address arbitrary programs and faults effectively. The abstraction-based MBSD approaches still suffer from spurious results, and the approaches relying on model checking either require a separate formal specification of the program behaviour, or may require excessive resources. (Semi-)Interactive incremental refinement using simple abstraction, histories about past executions and debugging sessions as well as existing design artifacts would be desired to simplify the specification of anticipated program behaviour, in particular for ad-hoc debugging scenarios and casual users not trained in formal methods.

We believe that the general debugging problem can only be solved by combination of different approaches. In this context, the MBSD approach has potential, with the MBSD engine as central component, unifying different debugging approaches though the common abstraction into components and fault modes, and its suitability for an iterative, interactive debugging session that reduces the number of diagnoses through multiple user queries (Wieland 2001).

Ongoing work aims at (i) broadening fault models to deal with a wider range of complex faults, such as assignments to incorrect variables, missing or swapped statements, etc., and (ii) providing simple user interaction for incremental specification of complex program behaviour. Ultimately, extensions to existing approaches capable of isolating complex faults and missing code are desired to cope with general programs.

References

Chaki, S.; Groce, A.; and Strichman, O. 2004. Explaining abstract counterexamples. In Taylor, R. N., and Dwyer, M. B., eds., *SIGSOFT FSE*, 73–82. ACM.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM* 50(5):752–794.

Cleve, H., and Zeller, A. 2005. Locating causes of program failures. In Roman, G.-C.; Griswold, W. G.; and Nuseibeh, B., eds., *ICSE*, 342–351. ACM.

Console, L.; Friedrich, G.; and Dupré, D. T. 1993. Model-based diagnosis meets error diagnosis in logic programs. In *Proc. 13th IJCAI*, 1494–1499.

Cousot, P., and Cousot, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *POPL'77*, 238–252.

Friedrich, G.; Stumptner, M.; and Wotawa, F. 1996. Model-based diagnosis of hardware designs. In Wahlster, W., ed., *ECAI*, 491–495. John Wiley and Sons, Chichester.

Griesmayer, A.; Staber, S.; and Bloem, R. 2006. Automated fault localization for C programs. In *V&D'06*, Electronic Lecture Notes in Theoretical Computer Science. Springer-Verlag.

Groce, A., and Visser, W. 2003. What went wrong: Explaining counterexamples. In Ball, T., and Rajamani, S. K., eds., *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, 121–135. Springer-Verlag.

Groce, A. 2004. Error explanation with distance metrics. In Jensen, K., and Podelski, A., eds., *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, 108–122. Springer.

Groce, A. D. 2005. *Error Explanation and Fault Localization with Distance Metrics*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University.

He, H., and Gupta, N. 2004. Automated debugging using path-based weakest preconditions. In Wermelinger, M., and Margaria, T., eds., *FASE*, volume 2984 of *Lecture Notes in Computer Science*, 267–280. Springer-Verlag.

Köb, D.; Chen, R.; and Wotawa, F. 2005. Abstract model refinement for model-based program debugging. In *Proc. DX'05*, 7–12.

Mateis, C.; Stumptner, M.; and Wotawa, F. 1999. Debugging of Java programs using a model-based approach. In *Proc. DX'99 Workshop*.

Mayer, W., and Stumptner, M. 2003. Model-based debugging using multiple abstract models. In *Proc. AADEBUG '03 Workshop*, 55–70.

Mayer, W., and Stumptner, M. 2004. Model-based debugging with high-level observations. In *IFIP International Conference on Intelligent Information Processing (ICIIP)*.

Mayer, W., and Stumptner, M. 2007. Model-based debugging - state of the art and future challenges. *ENTCS* 171.

Offutt, A. J.; Lee, A.; Rothermel, G.; Untch, R. H.; and Zapf, C. 1996. An experimental determination of sufficient mutant operators. *TOSEM* 5(2):99–118.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Renieris, M., and Reiss, S. P. 2003. Fault localization with nearest neighbor queries. In *Proceedings 18th International IEEE Conference on Automated Software Engineering*, 30–39. IEEE Computer Society.

Tip, F. 1995. A Survey of Program Slicing Techniques. *Journal of Programming Languages* 3(3):121–189.

Wieland, D. 2001. *Model-Based Debugging of Java Programs Using Dependencies*. Ph.D. Dissertation, Technische Universität Wien.

Wotawa, F. 2002. On the relationship between model-based debugging and program slicing. *Artificial Intelligence* 135(1-2):125–143.

Fault Diagnosis of Civil Engineering Structures using the Bond Graph Approach

Abbas Moustafa[†], Matthew Daigle[‡], Indranil Roychoudhury[‡], Chris Shantz[†], Gautam Biswas[‡], Sankaran Mahadevan[†], Xenofon Koutsoukos[‡]

[†]Department of Civil and Environmental Engineering, Vanderbilt University, Nashville TN 37235, USA

[‡]Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville TN 37235, USA

{abbas.m.moustafa, matthew.j.daigle, indranil.roychoudhury, chris.shantz, gautam.biswas, sankaran.mahadevan, xenofon.koutsoukos}@vanderbilt.edu

Abstract

This paper develops a fault diagnosis methodology for civil engineering structures based on the bond graph approach. The bond graph theory provides a modeling framework that includes parametric models of the physical system and the sensors. Structural faults are modeled as abrupt or gradual damage in structural components. Sensor faults are modeled as biases or drifts from true measurements. Fault detection uses a statistical method to identify significant deviations of measurements from nominal behavior of the structure. Fault isolation is carried out by comparing predicted effects of hypothesized faults with observed behavior of the structure. Numerical illustrations of fault diagnosis of a frame structure driven by time-varying loads are provided.

Introduction

Damage identification and health monitoring of civil engineering structures have received significant research attention worldwide (Doebling *et al*., 1998, Peeters & Roeck 2001, Chang *et al*., 2003). Diagnosing faulty behavior in these structures is crucial to their safe operation, and developing robust damage detection methodologies is a challenging problem. Damage detection schemes can be grouped into model-based and signal-driven methods. In model-based methods, the change in the system parameters, such as the structure's natural frequencies, mode shapes, and structural members' stiffness or damping coefficients are employed to characterize damage. Examples of model-based damage identification techniques include the ARX, ARMAX and the Least Squares models (Wang & Haldar 1994 and Kathuda *et al*., 2005). Signal-driven methods deal mainly with statistical and numerical analysis of measurement data (Jiang & Mahadevan 2005, 2007). Realistic scenarios for damage detection will combine methods for early detection of the occurrence of the damage and then isolating the fault to components, such as a beam or a column in a building. More advanced methods will also quantify the size of the damage. This allows for repair of the damaged component before catastrophic failure.

Bond graphs (BG) are an explicit topological modeling tool for capturing the common energy structure of systems (Rosenberg and Karnopp 1983). It is based on modeling energy flow between system components and inherently enforces continuity of power and conservation of energy. Bond graphs provide a systematic framework for building consistent and well constrained models of dynamic physical systems across multiple domains that include electrical, mechanical, hydraulic and thermal systems. The topological structure of BG models causality constraints that provide the mechanisms for effective and efficient fault diagnosis based on cause and effect analysis.

Our diagnosis algorithm is based on the TRANSCEND diagnosis framework (Mosterman & Biswas 1999). The diagnosis models, temporal causal graphs (TCGs), are derived systematically from BG models and provide the temporal and causal relations between deviant observations made on the system during its operation and hypothesized faults. Residuals are computed as deviations in measurements from nominal behavior. In an ideal or undamaged system the residuals should be zero. Nonzero residual values imply faults in the structural components or in the sensors. An abrupt fault is a sudden change in a system parameter (e.g., a sudden reduction in the stiffness or the damping of a structural member). Incipient faults result from slow variation in any of the system parameters over time that causes the system behavior to drift from its steady state (e.g., degradation or corrosion of steel bars over a long period of time). We assume faults are persistent.

This paper presents a methodology for applying bond graph theory to damage diagnosis for civil structures. The TRANSCEND algorithms are applied to a new domain, namely building structures. The paper focuses on detection and isolation of faults in structural components and sensors. For simplicity, we make the single fault assumption, but the methodology can be extended to multiple faults (Daigle *et al*. 2006).

Modeling Methodology

Model-based diagnosis methodologies require system models that accurately represent system dynamics and also link faulty behaviors to components of the model. We adopt the bond graph modeling framework that provides both these features and present a methodology for modeling multi-story building structures.

Bond graph models are constructed using the Fault Adaptive Control Technology (FACT) system (Manders *et al*. 2006) within the Generic Modeling Environment (GME) software developed at Vanderbilt University (GME-5 2005). MATLAB Simulink® models are automatically constructed from the bond graph models in GME using the tools described in (Roychoudhury *et al*. 2007).

Modeling Building Structures using Bond Graphs

Figure 1 shows a two-story frame structure. We first examine the effectiveness and accuracy of bond graph models of structures as compared to traditional structural dynamics modeling techniques. The equivalent lumped-mass model (two degree of freedom system) of the frame structure is shown in Figure 1(b). The lumped parameter elements of resistance (damping) (D_1, D_2), capacitance (stiffness) (k_1, k_2) and inductance (inertia) (m_1, m_2) are interconnected using energy conserving junctions producing the topological structure shown in Figure 2. Structural faults are represented by persistent, abrupt changes of the lumped parameter values. This bond graph represents the equations of motion of the frame structure in an implicit form. The bond graph is modular with each floor represented as a separate block. The blocks are connected through bond 5 to complete the frame structure. The sensors are also modeled as separate blocks for measuring displacements. Therefore, this modeling framework is easily extendible to *N*-story structures by duplicating the block of the second floor *N*-1 times.

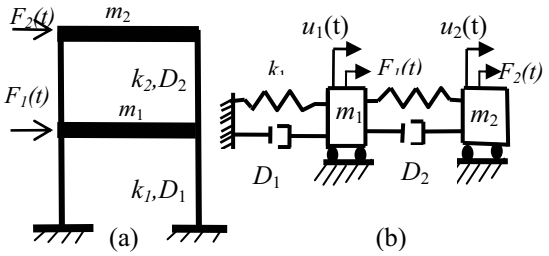


Figure1: (a) Frame building (b) Two-dof system

We first demonstrate the effectiveness of bond graphs to model these systems. To do this, we compute the dynamic responses of this system and compare them with those computed from the theory of structural dynamics. The structure is driven by the two sinusoidal forces $F_1(t) = 75\sin(9t)$ and $F_2(t) = 100\sin(9t)$ as shown in Figure 1. In our example, the stiffnesses of the structure are modeled by $k_1 = 30.70$ kN/m, $k_2 = 44.30$ kN/m, and the masses at the floor levels are $m_1 = 136$ N s^2/m, $m_2 = 66$ N s^2/m. The damping coefficients were modeled by the parameters $D_1 = 307$ Ns/m, $D_2 = 443$ Ns/m. The displacement, velocity and acceleration responses for the frame structure are computed using the derived simulation models and also by dynamic analysis using Duhamel's integral (Clough & Penzien 1993). Figure 3 depicts the displacement responses from the two methods. The figure shows a good match between responses computed using the two methods.

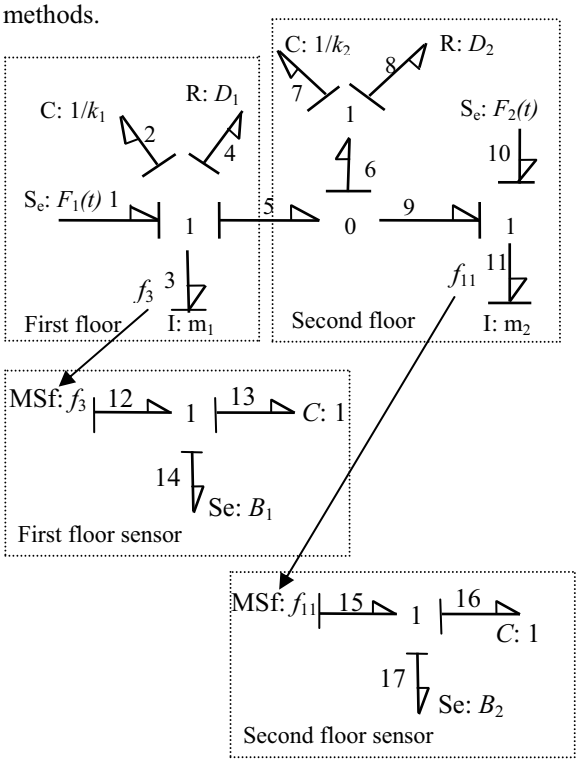


Figure 2: Bond graph model for the frame and the sensors

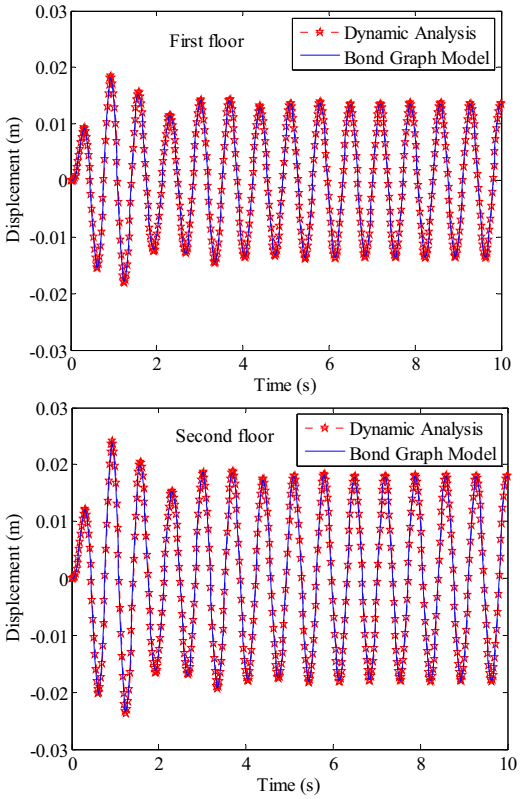


Figure 3: Dynamic response of the frame to sinusoidal loading

that the BG model provides sufficiently accurate results when compared to standard techniques from structural dynamics.

The bond graph model developed for the two-story frame structure was further examined when the structure is driven by the first horizontal component of the El Centro 1940 earthquake ground motion (SMDB 2000). The ground acceleration $\ddot{x}_g(t)$ is replaced by two horizontal forces $F_i(t) = -m_i \ddot{x}_g(t), i = 1,2$ acting on each floor. The displacement responses from the bond graph model and dynamic analysis is shown in Figure 4. Again, the structure responses show a good match compared with those obtained from the structural dynamics theory.

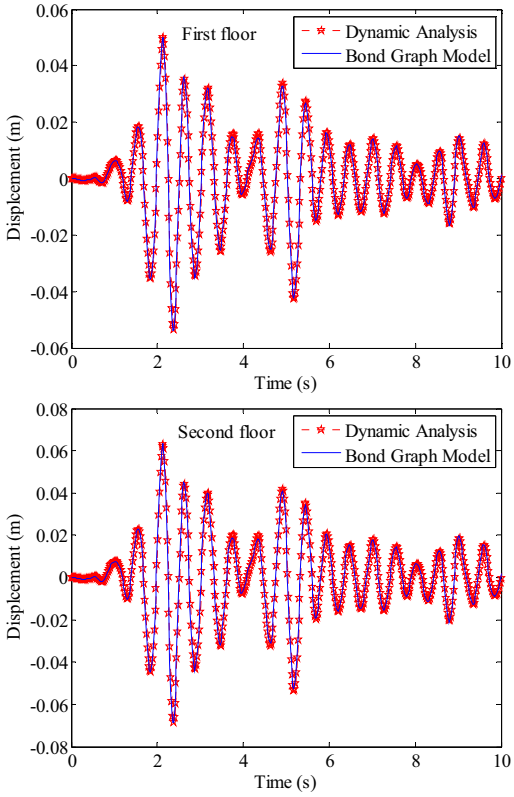


Figure 4: Dynamic response of the frame to earthquake acceleration

Modeling Sensors

From a practical point of view, deviations in measurements could result from damage in structural components or could also be attributed to sensor errors. For sensor errors, model-based identification models such as the finite element and the Least-Squares methods may not provide accurate results since these models do not account for sensor faults. On the other hand, bond graph models can easily model both the physical system and the sensors. Therefore, since faults are modeled as changes in the model parameters, bond graphs are capable of modeling faults in the structural components and sensors. This paper examines this issue by modeling sensors using bond graphs based upon the approach presented in (Daigle *et al*. 2006).

Faults in sensors could result in a bias or a drift in the measured values. Biased measurements are those measurements that deviate from their true value by a constant quantity *B*. Faults in sensor measurements could also result in a drift between the true value and the sensor measured value over time.

For a sensor measuring the displacement response, a bias fault implies that the actual measured displacement d_m is the sum of the true displacement d_t and the bias constant B. Nominally, B is zero. In the bond graph, the sensor is modeled using a modulated source of flow (MSf) and encapsulates the relation $d_m(t_i) = d_t(t_i) + B$, see Figure 2. As an example, consider the displacement sensor for the first floor. The effort associated with bond 12, e_{12} represents the measured displacement, the effort associated with bond 13 represents the true displacement, and the effort associated with bond 14 represents the bias quantity B. A C-element with $C = 1$ is included to integrate the true velocity to obtain true displacement. The effort balance of the 1-junction is $e_{12} = e_{13} + e_{14}$ where $e_{13}(t) = C\int f_{13}(\tau)d\tau = d_{13}(t)$ and $e_{14} = B_1$. This leads to $e_{12} = d_{13} + B_1$ or $d_m(t) = d_t(t) + B_1$.

In the case of drift, the measured response d_m is the sum of the true displacement d_t and the drift d_{drift}. The drift is a function that increases gradually with time. We model the drift using a linear function of time. Thus, $d_m(t) = d_t(t) + d_{dr}(t)$ or $d_m(t) = d_t(t) + gt$, where g is a constant representing the slope of the drift function. Again, the sensor is modeled in the same way as for bias, but the bias term is replaced by the drift function $d_{drift}(t) = g\,t$. Nominally, g is 0.

Fault Diagnosis Approach

The fault diagnosis methodology applied in this paper is a model-based method and consists of fault detection, symbol generation and qualitative isolation of faults. Figure 5 shows the overall architecture for the fault diagnosis approach. The parameter estimation of faulty components (fault identification) is not considered in the numerical examples. The details of the diagnosis procedures are provided below.

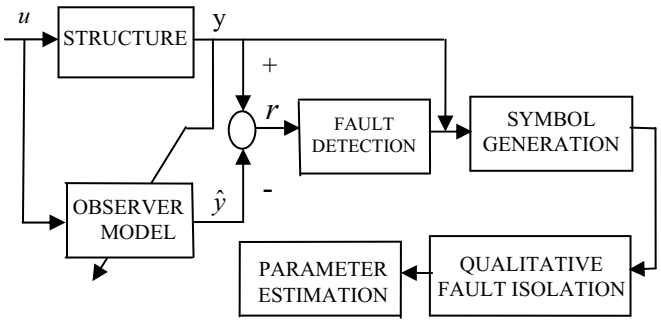


Figure 5: Diagnosis of dynamic systems

Fault Detection and Symbol Generation

The detection scheme is based on comparison of measurements from faulty and nominal structures. Herein, we distinguish between two distinct types of faults, namely, *abrupt* or sudden faults and *incipient* or degrading faults. Abrupt faults reflect a discontinuous reduction in any of the system parameters (e.g., a sudden reduction in the stiffness of a member of the structure). Incipient faults, on the other hand, represent a slow change in any of the model parameters. Abrupt faults can be expected to happen to structures subjected to short duration high amplitude loads such as blast loadings or strong motion earthquakes. Incipient faults could be expected to occur to structures in their steady state region such as structures subjected to vibrations from machineries.

The detection procedure is carried out by comparing the measurements of nominal (normal or undamaged) to faulty (damaged) measurements. This step leads to determining the residuals, the difference between observed and estimated behavior, which, when statistically significant, imply fault occurrence. The structure is considered to be faulty if the residual at any time point exceeds a predefined threshold value. When noise is included in the measurements, the Z-test is performed to examine if the structure is faulty or not.

From a practical and experimental point of view, the presence of noise in response measurements cannot be avoided. Noise implies uncertainty in the sensors which could be due to errors, inaccuracy, imprecision, or imperfection. When noise is present in the measurements of the structure responses, the estimation of the residual cannot be robustly carried out by simple subtraction of the data and comparing the residual with a predefined threshold value. Instead, a measure of the deviation of the residual from the ideal value (typically zero) is defined. The measure of the deviation at time t_i is defined by the average residual for the last N_2 measurement time points.

The decision whether the deviation is significant or not is based on a hypothesis test. In the present study we use the Z-test for this purpose (Biswas, *et al*., 2003). To perform the Z-test, the variance of the residual should be known. To approximate the conditions necessary for the Z-test, the variance of the signal is estimated but for a larger set of samples containing N_1 observation data points, where $N_1 >> N_2$. This approach is applied also to determine the slope of the deviation once detected.

The symbol generation follows the comparison of the measurements from nominal and faulty behavior. If the measurement of the faulty structure at a given point of time is above normal, a + symbol is assigned, and if below normal, a – symbol is assigned.

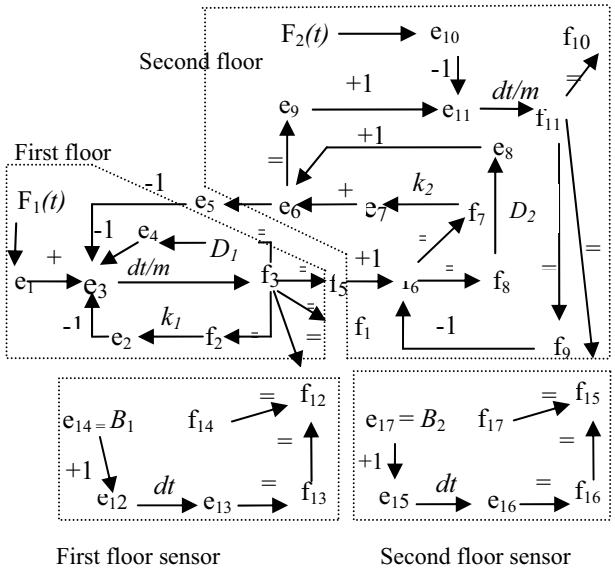


Figure 6: Temporal causal graph for the frame and the sensors

Fault Isolation

When faults are detected, fault isolation begins. The isolation scheme is based on the temporal causal graph (TCG), which is derived systematically from the bond graph model of the system (Mosterman & Biswas 1999). The TCG is derived based on the cause and effect relationships of the system variables and the constitutive equations of BG elements. For the two-story frame structure, the TCG is shown in Figure 6. The structure is first modeled using bond graphs and the TCG is constructed. The TCG links faults to their causal effects on measurements, called fault signatures. Fault signatures represent 0^{th} through k^{th} order derivative changes on a measurement residual at the point of fault occurrence. They provide the discriminatory power in the fault isolation approach.

The TCG is used to perform a backward propagation for the deviant measurement to generate the possible fault causes. This identifies a set of parameters that could be the reason for the fault. It also determines whether the cause is an increase or a decrease in the parameter. Some of these such as increase or decrease in the masses or increase in the stiffness or damping parameters are omitted. The qualitative fault isolation consists of two steps which are:

1. Perform forward propagation for each possible fault scenario determined in the previous step, to estimate the fault signature matrix. This matrix contains the qualitative effect of the change in the structure parameters (reduction - or increase +) on the measurements quantities. For instance, the signature of the stiffness parameter k_1 is derived from the TCG by assuming that k_1 decreases and tracking the effect of this on the displacements u_1 and u_2 and their derivatives. The signatures of other parameters are derived following the

same procedure. The fault signature for the frame structure of Figure 1 is provided in table 1.

2. Carry out progressive monitoring on each of these fault causes and the one that matches the fault signature is the actual fault.

Table 1: Fault signatures for the frame up to the first non zero direction of change

Fault	First floor displacement (u_1)	Second floor displacement (u_2)
k_1^-	00+	000+
k_2^-	00+	00-
D_1^-	00+	000+
D_2^-	00+	00-
B_1^+	+	0
B_2^+	0	+
B_1^-	-	0
B_2^-	0	-
dr_1^+	0+	00
dr_2^+	00	0+

The computed fault signatures of the frame structure are shown in Table 1. We only show the signature up to the first nonzero direction of change. Faults which produce discontinuities on the measurements (0^{th} order changes) provide additional discriminatory power. Higher order effects eventually manifest as first order effects, and since we can only measure magnitude and slope reliably, only the first change (and whether it was discontinuous) is useful. Since we measure displacement, the fault signatures represent a qualitative measure that reflects how a displacement observation could be affected by a change (reduction or increase) in one of the structure parameters. For instance, to derive the fault signature of the stiffness of the first floor k_1, a forward propagation is performed using the TCG of Figure 6 by tracing the qualitative effect of a decrease in k_1 on the response measurements u_1 and u_2. The signature is found to affect the second derivative of u_1 and the third derivative of u_2 and thus its signatures are 00+ and 000+, respectively. Signatures of other parameters are generated following the same procedure, described in detail in (Mosterman &Biswas 1999). It is observed from the table that the fault in the stiffness parameter of the first and the second floors have different signatures on the measurements u_1 and u_2. Furthermore, the signature of a fault in the stiffness parameter k_1 is the same due to a fault in the damping parameter D_1. The same observation is valid for the stiffness and damping coefficient of the second floor. This observation is consistent since the reduction in the stiffness or in the damping coefficient implies damage to the columns of the same floor, and in practical situations, knowing this is enough.

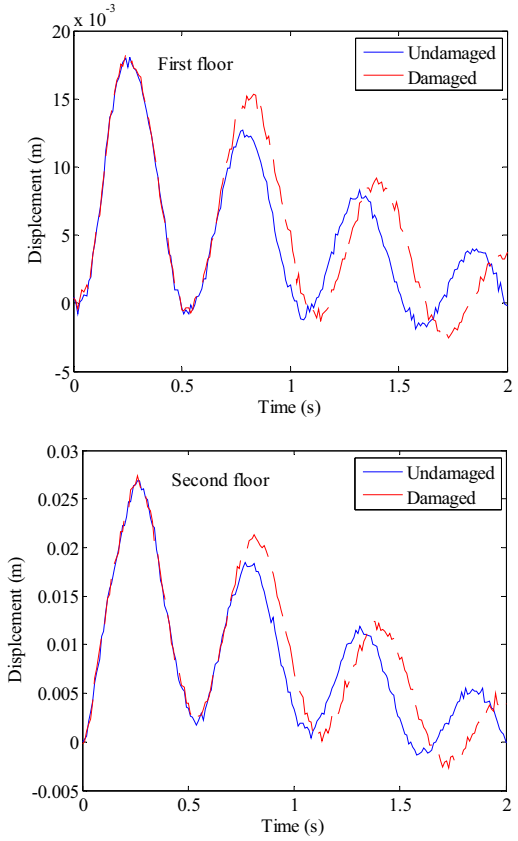


Figure 7: Noisy measurements (5% noise)

Numerical Examples

Two-story Building under Blast Loading

The two-story shear frame of Figure 1 is considered. This structure was studied by Wang & Haldar (1994) within the context of system identification of frame structures with unknown inputs. The structure is acted upon by a blast load of peak amplitude of 150 kN at t = 0 s decreasing linearly to zero at 2.0 s and applied at the second floor. The numerical values for stiffness, masses and damping considered previously are adopted in this example for simulating the theoretical response measurements for the undamaged and damaged structure. The displacement, velocity and acceleration responses at the two floors are computed using the simulation model generated from the bond graph. The possible fault causes considered include faults in the structural components (k_1, k_2, D_1, D_2) and bias (B_1, B_2) or drift (dr_1, dr_2) in the sensors.

Table 2: Natural frequencies of undamaged and damaged structure (Rad/s)

	Undamaged structure	Damaged structure (20 %)	
		k_1 damaged	k_2 damaged
ω_1	11.83	10.68 (9.72 %)	11.69 (1.18 %)
ω_2	32.91	32.61 (0.91 %)	29.78 (9.51 %)

The sampling time of the simulated responses was taken as 0.01 s. For the damaged structure, reductions of 10 % and 20 % in the parameters k_1, k_2, D_1 and D_2 were used for the experimental study. Table 1 summarizes the first two natural frequencies for the undamaged and the damaged structure due to damage of 20 % in the stiffness parameters. The table summarizes also the associated percentage reduction in natural frequencies. Figure 7 shows the theoretical displacement measurements for the undamaged and damaged structure.

Faults in Structural Components

We first consider damage in the columns of the first floor. The fault is simulated by reducing the stiffness parameter k_1 at t = 0.60 s by 20 % and the theoretical measurements u_1 and u_2 were computed using the simulation model. The noise free displacements of the undamaged and damaged structures were used as observations. Damage is considered to occur if the displacement measurements deviate by more than 5 % of the nominal values. The diagnosis model detects a fault at 0.70 s due to deviation in u_1 exceeding the threshold quantity adopted. The displacement u_2 deviates at 0.76 s. The backward propagation determines that the cause for the fault could be due to reduction in the stiffness parameters k_1 and k_2 or the damping coefficients D_1 and D_2 or due to increase in the masses m_1 *and* m_2. In this study, the possibility of faults occurring due to increases in the masses is eliminated since masses are assumed to remain unchanged.

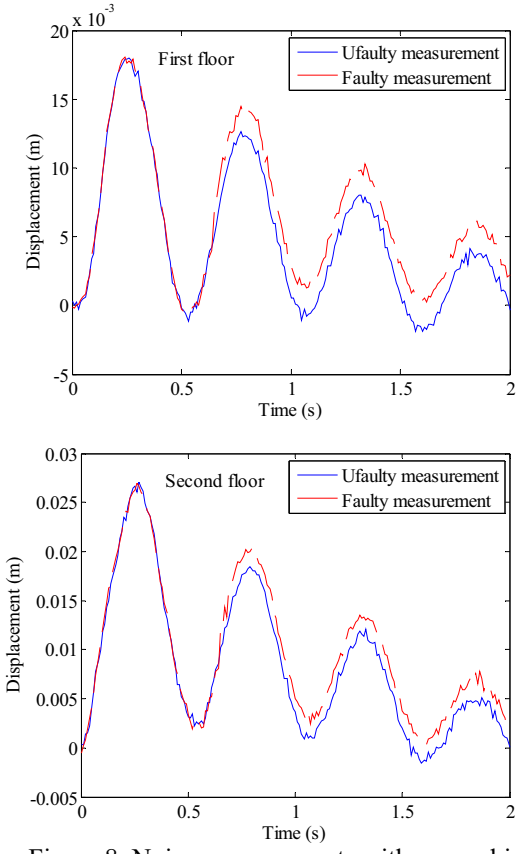


Figure 8: Noisy measurements with sensor bias

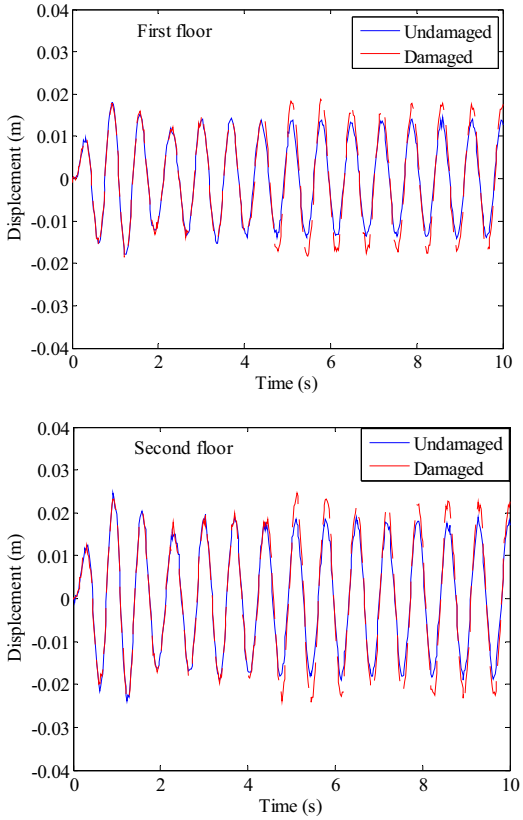


Figure 9: Noisy measurements (abrupt fault)

Table 3: Diagnosis results for the frame structure

Fault	Injection time (s)	Detection and diagnosis time
k_1^-	0.60 (20%)	u_1 deviates, 0.73, u_2 deviates 0.76 k_1^- or D_1^-
k_2^-	0.60 (20%)	u_2 deviates, 0.68, u_1 deviates 0.70 k_2^- or D_2^-
D_1^-	0.55 (20%)	u_2 deviates, 0.89, u_1 deviates 0.91 k_1^- or D_1^-
D_2^-	0.55 (20%)	u_2 deviates, 0.88, u_1 deviates 1.15 k_2^- or D_2^-
B_1^+	0.65 (0.002 m)	u_1 deviates, 0.68
B_2^+	0.65 (0.002 m)	u_2 deviates, 0.68

The forward propagations for reduction in the parameters k_1 or D_1 lead to the second derivatives of u_1 and u_2 above normal (00+). Similarly, the signature of reduction in k_2 or D_2 is accompanied by $\ddot{u}_1$ below normal (00-) and $\ddot{u}_2$ above normal (00+), see table 1. Comparing these signatures with the fault signature in Table 1, the fault is

identified in the columns of the first floor (due to reduction in either k_1 or D_1). A similar analysis with a reduction of 20 % in the damping coefficient D_1 was also carried out and the progressive monitoring leads to identifying the damage in the columns of the first floor. It is to be noted that the diagnosis model successfully isolates the fault to be in the columns of the first floor or the second floor. From engineering point of view, this is considered to be sufficient since damage in the columns implies reductions in stiffness and damping. The inspection that follows this step determines the location of the damage and necessary repair.

To address the issue of noise in measurements, a numerically generated stationary Gaussian white noise with zero mean and intensity 5 % of the root-mean-square values of the displacement at first floor is added to the theoretical responses. With the noise included in the response data, the structure is again diagnosed and some of these results are in Table 3 and Figures. 7-8. Herein, the Z-test is employed with the mean residual adopted in detecting the fault. In the numerical analyses, the parameters N_1, N_2, α are taken as 50, 5 and 1.0, respectively. By injecting a fault of 20 % reduction in the stiffness parameter k_1 and for a noise level of 5 %, the fault was detected at 0.73 s due to deviation on u_1 and u_2 deviates at 0.76 s (for noise free the fault was detected at 0.70 s). This implies that the presence of noise in the measurements may result in predicting the fault at a different time compared with the case of noise free measurements. The diagnosis model successfully detects and isolates the fault source to be in the stiffness or the damping coefficient of the first floor.

Faults in Sensors

We consider sensor faults modeled as bias in the measurements. We inject a bias of 0.002 m to the displacement at the first floor at 0.65 s. The true and faulty measurements at the first floor with noise included are shown in Figure 8. A similar analysis was carried out for a bias in the second sensor, see Figure 8. The fault signatures for individual biased measurements B_1 and B_2 are provided in Table 1. The signature of the bias in one sensor on the displacement measured by the same sensor is +. The bias or fault in one of the sensors has no effect on the other sensor which is physically true. In bond graph setting there is no path that connects the element source that simulates the bias element in one sensor (e.g, $S_e : B_1$) and the measurement in the other sensor (f_{13}) and therefore the signature is 0. For the bias B_1, the diagnosis model successfully detects the fault to be in the first sensor at 0.68 s. For the noise free measurements, the fault was identified at 0.65 s. A similar analysis with bias in the second sensor was carried out and the diagnosis model successfully detects the fault to be in the second sensor, see Table 3.

Two-story Building under Sinusoidal Load

To examine the applicability of the diagnosis model developed in this paper to alternative external loading conditions we reconsider the frame structure of Figure 1. The structure is assumed to be subjected to the sinusoidal forces $F_1(t) = 75\sin(9t)$ and $F_2(t) = 100\sin(9t)$ as shown in Figure 1(a). The frequency of the driving forces is selected to be close to the first natural frequency of the undamaged structure. The total duration of the input is taken as 10.0 s. In the previous example, the damage was introduced by decreasing the structure stiffness or damping at a certain time by some value. In practical situations, it is possible that damage to the structural members may occur gradually over time and not suddenly. For this reason, this example examines modeling degrading faults. This aspect, to the best of our knowledge has not been considered earlier within the context of fault diagnosis of civil structures.

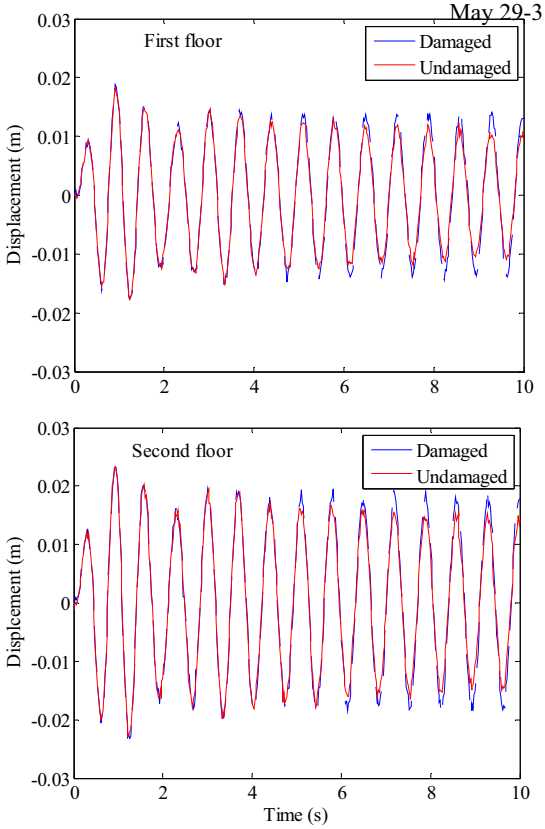


Figure 10: Noise free measurements (degrading fault)

We consider abrupt faults first. A reduction of 10 % in the stiffness parameter k_1 is applied at $t = 4.0$ s and the simulation model is used to generate the displacement measurements at the floors levels. Figure 9 shows the noisy displacement measurements for the undamaged and the damaged structure. The damage was detected at 4.10 s for the noise free measurements and at 4.16 s for the noisy measurements due to deviation in u_1. The numerical values for N_1, N_2 and α were taken as in the previous Example. Comparing the fault signatures of Table 1 and the measurements variation, it is seen that the damaged parameter is either k_1 or D_1 . Similar analysis was also obtained when the damage was introduced to the stiffness of the second floor and the diagnosis model detects the fault and isolates k_2 or D_2 as the damaged cause.

In the above analysis, the structure damage was modeled by a sudden reduction of 10 % in the stiffness parame-

ter k_1 at $t = 4.0$ s. To model a degrading damage, i.e., slowly varying damage to the structure, we introduce a linear reduction to the stiffness of the first floor. The damage represents a uniform reduction of 15 % in the parameter k_1 over the time period 3.0 to 10.0 s. Again, the structure displacement response for the damaged structure was simulated using the simulation model (see Figure 10). The diagnosis algorithm detects the damage at $t = 5.32$ s and the defect is in the columns of the first floor.

Conclusions

This paper develops a fault diagnosis methodology for civil engineering structures based on a qualitative approach to fault isolation. The paper focuses on fault detection and qualitative isolation of defected structural and sensor components. The proposed diagnosis scheme models sensors using bond graph components and therefore is capable of detecting and isolating faults in sensors as well. In this context, it may be emphasized that alternative damage diagnosis techniques such as ARX, ARMAX and the least-squares models do not account for sensor faults.

For verification purposes, both noise-free and noisy response measurements were considered. The advantage of the present diagnosis scheme is the isolation of faulty components with less computation compared to solely quantitative diagnosis methods such as ARX, ARMAX and the least-squares approach. In the present paper, fault diagnosis of a two-story frame structure was studied. The diagnosis of larger and complex structures is of interest. Furthermore, the diagnosis algorithm can be further improved by including a least-squares scheme to quantify the fault size of the isolated faulty component.

In the present study, physical dynamic systems such as frame structures were modeled using shear frame models. The use of more accurate models for continuous systems taking into account translational and rotational degrees of freedom is of significant interest. It is also of interest to investigate multiple fault scenarios.

References

Biswas, G., Simon, G., Mahadevan, N., Narasimhan, S., Ramirez, J., Karsai, G., 2003, A robust method for hybrid diagnosis of complex systems, In Proc. of the 5th Symposium on Fault Detection, Supervision and Safety for Technical Processes, pages 1125-1131, Washington, DC, June 2003.

Chang, P.C., Flatau, A., Liu, S.C., 2003, Review paper: Health monitoring of civil infrastructure, *Structural Health monitoring*, 2:3, 257-267.

Clough, R.W., Penzien, J., 1993, *Dynamics of structures*, McGraw-Hill, 2nd Edition, Tokyo.

Daigle, M., Koutsoukos, X., Biswas, G., 2006, Distributed diagnosis of coupled mobile robots, *2006 IEEE International Conference on Robotics and Automation*, 3787-3794.

Daigle, M., Koutsoukos, X., Biswas, G., 2006, Multiple fault diagnosis in complex physical systems, In *Proceedings of the 17th International Workshop on Principles of Diagnosis,*, 69-76.

Doebling, S.W., Farrar, C.R., Prime, M.B., 1998, A summary review of vibration-based damage identification methods, *The shock and vibration digest*, 30(2), 91-105.

GME-5 2005, A Generic Modeling Environment, GME 5 User's Manual, Vanderbilt University, http://www.isis.vanderbilt.edu.

Jiang, X., Adeli, H., 2007, Pseudospectra, MUSIC and dynamic wavelet neural network for damage detection of high-rise buildings, *International Journal for Numerical Methods in Engineering*, in press.

Jiang, X., Mahadevan, S., Adeli, H., 2007, Bayesian wavelet packet denoising for structural system identification, *Structural Control and Health Monitoring*, in press.

Katkhuda, H, Martinez, R., Haldar, A., 2005, Health assessment at local level with unknown input excitation, *Journal of Structural Engineering*, 131(6), 956-965.

Manders, E. J.; Biswas, G.; Mahadevan, N.; and Karsai, G 2006. Component-oriented modeling of hybrid dynamic systems using the Generic Modeling Environment. In *Proceedings of the 4th Workshop on Model-Based Development of Computer Based Systems*.

Mosterman, P.J., Biswas, G., 1999, Diagnosis of continuous valued systems in transient operating regions, *IEEE Transactions, Man, and Cybernetics-Part A*, 29(6).

Peeters, B., Roeck, G.D., 2001, Stochastic system identification for operational modal analysis: A review, *Journal of Dynamic Systems, Measurements, and control*, 123, 659-667.

Rosenberg R.C., and Karnopp D.C., 1983, *Introduction to physical system dynamics*: McGraw-Hill, New York.

Roychoudhury, I.; Daigle, M.; Biswas, G.; Koutsoukos, X.; and Mosterman, P. J. 2007. A Method for Efficient Simulation of Hybrid Bond Graphs. In *Proceedings of the International Conference on Bond Graph Modeling (ICBGM 2007)*,177-184.

SMDB, 2000, The Strong Motion Data-based, http://smdb.crustal.ucsb.edu, South California Earthquake center.

Wang D., Haldar, A., 1994, Element-level system identification with unknown input, *Journal of Engineering Mechanics*, 120(1), 159-176.

Using An Oriented J-Measure to Prune Chronicle Models

Benayadi Nabil and **Le Goc Marc**
LSIS, UMR CNRS 6168, Université Paul Cézanne
Domaine Universitaire St Jérôme
13397 Marseille cedex 20, France
nabil.benayadi, marc.legoc@lsis.org

Abstract

This paper address the problem of discovering chronicles models from the timed data contained in the data base of a knowledge based system that monitors, diagnoses and controls a blast furnace. The key idea is to add an entropic criterion in the Stochastic Approach, which is based on the representation of sequences of discrete event class occurrences as a homogenous Markov chain and its corresponding superposition of Poisson processes. This representation allows an abductive reasoning to discover temporal knowledge and to represent it as an abstract chronicle model tree (the BJT4T algorithm). We define the BJ-Measure of a sequential binary relation as an adaptation of the J-measure of the Information Theory between two variables. The design of the BJ-Measure has lead first to modify the BJT4T algorithm to reject the relations when the antecedent and the consequent are independent. And second, the BJ-Measure has lead to define an heuristic to prune the BJT4T trees with the aim of reducing the search space of potential diagnosis chronicle models. The first results of this approach are presented on the alarms of the SACHEM system of the Arcelor Mittal Steel group.

Introduction

The aim of our works is to define a method for discovering the relations between alarms in order to predict and avoid undesirable alarms. The problem of discovering the relations between alarms in a sequence of alarm occurrences can be formulated as follow: given a sequence of alarm occurrences, what is (are) the model(s) that allows predicting the occurrences of an alarm? This problem is still an open problem (Mannila 2002) and one of the difficulties is the combination of logical relations and temporal constraints ((Cauvin *et al.* 1998), (Hanks & Dermott 1994)). This problem is similar to the "chronicle learning" problem where chronicle models are frequent "patterns" of alarms. A chronicle model is a graph where the nodes are alarms and the arcs are timed constraints between the alarms occurrences (Ghallab 1996). This formalism can be used to represent the frequent patterns that are discovered with algorithms inspired from the Data Mining domain (cf. ((Cordier & Dousson 2000), (Dousson & Duong 1999) for examples in a telecommunication network). (Le Goc & Bouché 2005) proposes a stochastic approach to model a sequence of discrete event class occurrences to avoid the main problems of the frequency approach of the Data Mining domain (see next section). The stochastic approach considers that a sequence of discrete event class occurrences is a particular execution of a timed stochastic automaton. A sequence can then be represented in the dual forms of a homogeneous Markov Chain and its superposition of Poisson processes, which approximates the stochastic and the temporal properties of the corresponding timed stochastic automaton. Timed relations between discrete event classes can then be directly deduced from these representations.

The next section presents a brief state of art of the various approaches for discovering temporal patterns from the timed data contained in a data base. Section 3 introduces the stochastic approach, and section 4 presents the BJ-Measure that allows to prune the chronicle models produced with the stochastic approach . An application of this approach on Sachem system's alarms is proposed in section 4 and the section 5 concludes the paper on our current works.

Related Works

The Data Mining domain aims at defining techniques to find a minimal set of relations of the type $y \rightarrow x$ that characterize a database (Agrawal, Imielinski, & Swami 1993). Generally speaking, these techniques generate a large amount of rules, much of them being redundant and not informative according an application domain, and largely differ according to the fact that the data are timed or not.

Mining a Data Set

A lot of measures have been defined to evaluate the interest of the discovered relations. According to (Blanchard *et al.* 2004), the interest notion is concerned the "generality" of an n-ary relation (the number of examples), evaluated with its "support" (Agrawal, Imielinski, & Swami 1993), (Kodratoff 2000), the "implicative power" (i.e. the ability of a relation $y \rightarrow x$ to deduce an x given a y) evaluated with the Confidence Measure (Agrawal, Imielinski, & Swami 1993), the Inclusion Rate (Gras *et al.* 2004), the Sebag-Schoenauer Measure (Sebag & Schoenauer 1988), the Loevinger Measure (Loevinger 1947) or the J-Measure (Smyth & Goodman 1992), or the "Statistical Significant", evaluated with

the χ^2 test (Brin, Motwani, & Silverstein 1997), the Intensity implication (Gras 1996) or its Entropic version (Gras *et al.* 2001).

Let be X and Y two random variables taking a value in the respective sets $X = \{x_i\}, i = 0,1,\ldots,n$ and $Y = \{y_j\}, j = 0,1,\ldots,m$. Considering the assignation of a particular value x_i chosen in a set of n values to the variable X as a discrete event occurrence (idem for Y), Shannon defines the entropy of the joint event $(Y = y_j, X = x_i)$ as the quantity $H(X,Y)$ (Shannon & Weaver 1949):

$$H(Y,X) = \sum_{j,i} p(j,i) \times \log(\frac{1}{p(j,i)}) \quad (1)$$

where $p(j,i) \equiv p((Y = y_j, X = x_i))$. Since $p(Y = y_j) \equiv p(j) = \sum_i p(j,i)$, we have:

$$H(Y) = \sum_{j,i} p(j,i) \times \log(\sum_i \frac{1}{p(j,i)}) \quad (2)$$

Considering a Markovian process that generates a sequence of events $X = x_i$ and $Y = y_j$ according to the probabilities $p(i)$ and $p(j)$ of the values x_i and y_j, Shannon defines the conditional entropy of X, $H_{Y=y_j}(X) \equiv H(X|Y = y_j)$ as the average of the entropy of X for each value of Y, weighted according to the probability of Y of getting the particular value y_j (Clark & Niblett 1989):

$$H(X|Y = y_j) = \sum_i p(j,i) \times \log(\frac{1}{p(i|j)}) \quad (3)$$

where $p(i|j)$ is the probability of the transition from the event $Y = y_j$ to the event $X = x_i$ in the considered Markovian process. By definition, $p(j,i) = p(i|j) \times p(j)$ and $\sum_j p(i|j) = 1$.

From these definitions, different measures have been defined in order to evaluate the link between variables. Generally speaking, an Information Theory based measure of the information exchanged between two variables Y and X is an evaluation of the information provided by the event $Y = y_j$ on the values of a variable X whenever the event $Y = y_j$ changes the prior probabilities $\{p(i)\}$ of the various possible values x_i of X to a new set of $p\{i|j\}$ posterior probabilities (Blachman 1968). So, any measure of the form $F(X;y_j) = f(P,Q)$ linked with the information entropy of a random variable X that take into account the prior $P(X)$ and the posterior $Q(X)$ probabilities of X is an Information Theory based measure that can used to reason on the relations between a given set of variables. For example, the Mutual Information, also called the Entropy diminution or the Entropy gain, measures the average amount of the information shared between two variables Y and X (Jaroszewicz & Simovici 2001):

$$I(Y,X) = H(X) - \sum_j H(X|Y = y_j) \quad (4)$$

The Theil Uncertainty Coefficient measures the entropy reduction rate of X due to Y (Theil 1970):

$$\mu(Y,X) = \frac{I(Y,X)}{H(X)} \quad (5)$$

The J-Measure represents the mutual information amount shared between the variable X and the value y_j (Smyth & Goodman 1992):

$$\begin{aligned} J(X,Y = y_j) &= p(j) \times \sum_i p(i|j) \times \log(\frac{p(i|j)}{p(i)}) \\ &\equiv p(j) \times j(X,Y = y_j) \end{aligned} \quad (6)$$

The J-Measure is unique, never negative and null at the independence point (Blachman 1968). According to (Smyth & Goodman 1992), the J-Measure is a combination of two terms (equation 6): (i) $p(j) \equiv p(Y = y_j)$ representing the probability of the event $Y = y_j$, and (ii) $j(X,Y = y_j)$ representing the Cross-Entropy between the distribution of the probabilities of the value of X when ignoring the event $Y = y_j$ and the distribution of the conditional probabilities of the values of X considering the event $Y = y_j$ (Shore & Johnson 1980). In other words, $j(X,Y = y_j)$ is a measure of the distance between the two distributions (Figure 1).

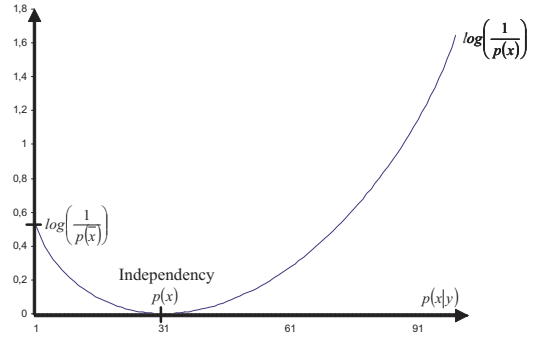


Figure 1: Graphical Representation of $j(X,Y = y_j)$

Mining a Data Sequence

When the data are timed, the data base is a sequence, a series of symbols ordered with an index more or less explicitly connected with a set of times. Two basic problems occurs when mining a sequence: (i) the relations depends of the order of the symbols (sequentiality) and (ii) symbols are not directly related with variables. The first problem entails some difficulties to adapt the Information-Theory measures that have proved to be efficient when mining data sets (Blanchard *et al.* 2005). To solve the the second one, Blanchard and al. (Blanchard *et al.* 2005) propose an extended notion of the Mutual Information, the Directed Information Ratio, but very few applications have been yet published.

Agrawal and al. (Agrawal & Srikant 1995) proposes another type of approach to discover sequential relations from

a large sequence. The method consists in finding a suite of items that is frequently observed in the sequence. The frequency is the number of times a pattern is observed in a given sequence (i.e. its support). This approach, called the Frequential Approach, is the basic of a set of algorithms like APRIORIALL, APRIORISOME and DYNAMICSOME. Mannila and al. (Mannila, Toivonen, & Verkamo 1997) propose another method to discover sequential patterns, called *episodes*, in a sequence of discrete events corresponding to the alarms of a telecommunication network (Hatonen *et al.* 1996a; 1996b). An episode is a collection of events that appear relatively close between them in a partial order. The episode having a frequency over a minimal threshold is then considered as a sequential pattern (WINEPI and MINEPI algorithms).

In the temporal logic domain, Ghallab (Ghallab 1996) proposes the notion of chronicle model to represent a set of timed binary relations between events. A chronicle model is a kind of temporal pattern specification where nodes are events and links are timed binary constraints represented as [min, max] intervals. Gallab's method for discovering chronicle models consists in splitting a set of sequences in examples and counter examples and finding the longest patterns that are common to the examples and that are not included in the counter examples. The main problem of this method is precisely to define what is an example and what is a counter example. This explains why, with the "FACE" algorithm, Dousson and Duong (Dousson & Duong 1999) has adapted the notion of chronicle models to the method of (Agrawal & Srikant 1995). But no sound method is provided to evaluate the timed constraints of the chronicle models.

The main problems of the frequency approach for mining a sequence are then the followings:

- The frequency approach fails at providing a global description of a sequence (Mannila 2002). This approach generates a large amount of relations that are representative of the recurrent parts of a data set, but not of the process that has generated the data set.
- The resulting n-ary relations are not linked with a specific behavior. Patterns being basically associations, it is often difficult to provide a semantic to the resulting relations.
- There is no sound theoretical basis for the evaluation of the timed constraints between the sequential pattern items.
- The computation time of the frequency approach is linked with the size of the patterns and the size of the sequence.

Principles of the Stochastic Approach

The Stochastic Approach aims at solving these problems when considering a given sequence $\omega = (o_k :: C^i), k \in K = \{0,\ldots,m-1\}$ of m occurrences o_k of discrete event classes C^i as the observable effects of a series of transition state of a timed stochastic automata (Le Goc & Bouché 2005), (Bouché, Le Goc, & Giambiasi 2005).

When the discrete event class occurrences are independent, the ω sequence can be represented under the form of a homogeneous Markov chain $X = (X(t_k); k \in K)$. To this aim, the set of discrete event classes $C_\omega = \{C^i\}_{i=0\ldots n-1}$ of ω is confused with the state space $Q = \{i\}_{i=0\ldots n-1}$ of the Markov chain X. A binary subsequence $\omega' = (\ o_{k-1} :: C^i, o_k :: C^j\ ) \subseteq \omega$ corresponds then to a state transition in $X : X(d(o_{k-1})) = i \rightarrow X(d(o_k)) = j$, where d is a function returning the occurrence time ($\forall o_k \equiv (t_k, C^i)\ d(o_k) = t_k$). X being homogeneous, the transition probability from a state i to a state j is a constant p_{ij}:

$$\begin{array}{l} \forall k \in K, P[X(t_k) = j | X(t_{k-1}) = i] = P[j|i] \equiv p_{ij} \\ p_{ij} = P\left[(o_{k-1} :: C^i, o_k :: C^j) \subseteq \omega | o_{k-1} :: C^i\right] \\ p_{ij} \equiv P\left[C^j | C^i\right] \end{array} \quad (7)$$

A timed sequential binary relation of the form $R(C^i, C^j, [\tau^-, \tau^+])$ can be constituted with a sequential relation $R_s(C^i, C^j)$ deduced from the transition probability matrix $P = [p_{ij}]$, the timed constraints $[\tau^-, \tau^+]$ being computed from the corresponding Poisson processes superposition. To this aim, the P matrix is weighted in a $B = [b_{ij}]$ matrix with the probability of the sub-sequence $(o_{k-1} :: C^i, o_k :: C^j)$ in ω:

$$\begin{array}{rcl} b_{ij} & = & p_{ij} \times P[(o_{k-1} :: C^i, o_k :: C^j) \subseteq \omega] \\ & = & P\left[(o_{k-1} :: C^i, o_k :: C^j) \subseteq \omega | o_{k-1} :: C^i\right] \\ & \times & P[(o_{k-1} :: C^i, o_k :: C^j) \subseteq \omega] \end{array} \quad (8)$$

Given a set $\Omega = \{\omega_i\}$ of sequences, the role of the "BJT" algorithm (Backward Jump with Timed constraints (Le Goc & Bouché 2005), (Bouché, Le Goc, & Giambiasi 2005)) is to compute the B matrix and the Poisson process superposition associated with Ω. Given a maximum depth and a maximum width, the "BJT4T" algorithm (BJT for Tree) uses these representations to constitute a set $M = \{R(C^i, C^j, [\tau^-, \tau^+])\}$ of the most probable timed binary sequential relations leading to a specific output discrete event class C^k (i.e. M constitutes a tree).

A path is a series $M = \{\ R(C^i, C^{i+1}, [\tau_i^-, \tau_i^+])\ \}$, $i = 0\ldots n$, of n timed sequential binary relations. The anticipation rate of a path M is the ratio between the number $i(M)$ of instances of M and the number $i(M')$ of instances where M' is the path M minus the last timed binary relation (i.e. $R(C^{n-1}, C^n, [\tau_{n-1}^-, \tau_{n-1}^+])$). An instance ω_m of M is a sub-sequence of $\omega_i \subset \Omega$ the occurrences of which are consistent with the logical and the timed constraints of M. For example, given a path $M_{123} = \{ R_{12}(C^1, C^2, [\tau_{12}^-, \tau_{12}^+])$, if $i(M_{123}) = 10$, and $i(M_{12}) = 15$, $M_{12} = M_{123} - R_{23}(C^2, C^3, [\tau_{23}^-, \tau_{23}^+])$, then the anticipation ratio of M_{123} is equal to $10/15 = 66\%$. The cover rate of a path is the ratio between the number of instances $i(M)$ and the number of occurrences of the final (output) class of M in Ω. For example, if $i(M_{123}) = 10$ and the number of discrete event class occurrence $o :: C^3$ in Ω is 20, the cover rate of M is equal to $10/20 = 50\%$. The Cover Rate is a kind of timed version of the support notion of the Frequential Approach. The "BJT4S" algorithm (BJT for Signatures) looks for the anticipating and the cover rates of each paths of such a tree.

The first problem with this method is that the number of paths contained in a tree is exponential with the width of

the tree. The second problem is that the complexity of the BJT4S algorithm is linear with the number of occurrences in Ω. So there is a need to define a way to prune such a tree in order to keep only sub branches containing strong relations between the classes. The Stochastic Approach defining a mathematical framework that is compatible with the one of Shannon's Information Theory, we propose then to define an oriented J-Measure, called the "BJ-Measure", to evaluate the quantity of information flowing from a discrete event class to another. This measure aims at providing a mean to solve the problem of Ghallab's method of splitting a set of sequences in examples and counter-examples. This paper presents an application of the BJ-Measure to prune a tree of timed binary relations provided by the Stochastic Approach.

The BJ-Measure

In the following, to avoid any confusion with the association symbol "$\rightarrow$", we will use the symbol "$\mapsto$" to denote a timed sequential binary relation: $R(C^i, C^o, [\tau^-, \tau^+]) \equiv C^i \mapsto C^o$.

Definition of the BJ-Measure

Considering the memoryless property of a Markov chain, the relation $C^i \mapsto C^o$ between the classes C^i and C^o can be view like one of the four relations linking the values of two random binary variables $Y = \{C^i, \neg C^i\}$ and $X = \{C^o, \neg C^o\}$ connected through a discrete memoryless channel (Figure 2, (Shannon & Weaver 1949)). This means that $\neg C^i = C_\omega - \{C^i\}$ and $\neg C^o = C_\omega - \{C^o\}$, so that $p(C^o|C^i i) + p(\neg C^o|C^i) = 1$. The basic definition of Shannon's condition information entropy (cf. equation 3) is then directly applied:

$$\begin{aligned} H(\{C^o, \neg C^o\}|C^i) &= -p(C^i) \times p(C^o|C^i) \times \log(p(C^o|C^i)) \\ &- p(C^i) \times p(\neg C^o|C^i) \times \log(p(\neg C^o|C^i)) \\ &= H(C^o|C^i) + H(\neg C^o|C^i) \end{aligned} \tag{9}$$

Consequently, an oriented cross-entropy based measure can be defined on a relation $C^i \mapsto C^o$.

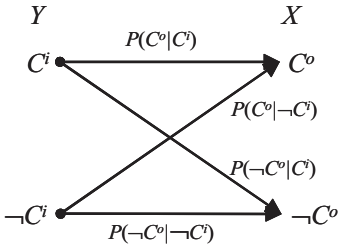


Figure 2: Discrete Event Memoryless Channel

Definition 1 *The occurrences of the event class C^i bring a quantity of the information to the occurrences of C^o in a given sequence ω if and only if $p(C^o|C^i) > p(C^o)$.*

Indeed, when $p(C^o|C^i) < p(C^o)$, the occurrences of C^i don't carry any information about the occurrences of C^o, and when $p(C^o|C^i) = p(C^o)$, the occurrences of the classes C^i and C^o are independent.

Definition 2 *Considering a timed sequential binary relation $C^i \mapsto C^o$ such that $p(C^o|C^i) > p(C^o)$, the BJ-Measure of $C^i \mapsto C^o$ is the cross-entropy between the occurrences of the C^i class and the occurrences of the set of class $\{ C^o, \neg C^o \}$. $BJM(C^i \mapsto C^o)$ is given by the following formula, where $n(C_\omega)$ is the number of discrete event classes with at least one occurrence in the sequence ω:*

$$\begin{aligned} &BJM(C^i \mapsto C^o) = \\ &p(C^o|C^i) \times \log(\tfrac{p(C^o|C^i)}{p(C^o)}) \\ &+ \tfrac{(1-p(C^o|C^i))}{N(C_\omega)-1} \times \log(\tfrac{(1-p(C^o|C^i))}{1-p(C^o)}) \end{aligned} \tag{10}$$

where $1 - p(C^o|C^i) = p(\neg C^o|C^i) = \sum_{k=0...n(C_\omega), k \neq o}(p(C^k|C^i))$.

By definition :

- if $p(C^i \mapsto C^o|C^i) < p(C^o)$, C^i don't carry information about C^o. The BJ-Measure must equal to 0.
- if $p(C^i \mapsto C^o|C^i) = p(C^o)$, C^i and C^o are independent and $BJM(C^i \mapsto C^o) = 0$.
- if $p(C^i \mapsto C^o|C^i) > p(C^o)$, then $BJM(C^i \mapsto C^o) > 0$.

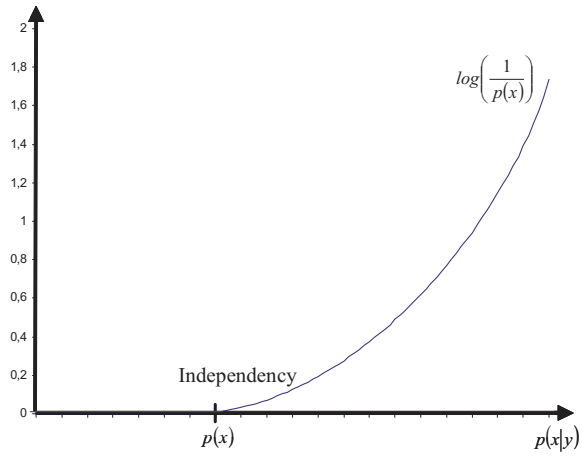


Figure 3: Form of the BJ-Measure

When $N(C_\omega) = 2$, the BJ-Measure behaves exactly like the right part of J-Measure (Figure 3). When $N(C_\omega) = 2$, the BJ-Measure evolves according the average distribution of probabilities of the transitions in the Markov chain. For example, given a probability $p(C^o) = 0.3$, the BJ-Measure increases according to the number $N(C_\omega)$ of classes in the sequence ω and the probability $p(C^o|C^i)$ (Figure 4).

The BJ-Measure is then an Information Theory based measure that is adapted to a Markov chain representing a sequence of discrete event class occurrences. This measure is then appropriate to estimate the quantity of information flowing in a sequential relation linking two discrete event classes.

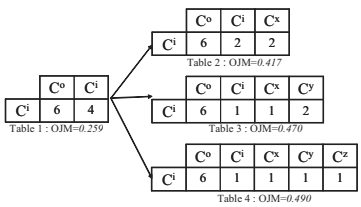


Figure 4: Example

The BJ-Measure of a path

The memoryless property of a Markov chain allows us to extend the BJ-Measure to an n-ary relation in the frame of the Stochastic Approach. Let us consider a set of timed sequential binary relations $S = \{(C^i \mapsto C^o)\}$ build from a sequence ω with the BJT4T algorithm. According to the memoryless property of a Markov chain, the binary relations contained in S are independent.

Any subset of S of the series form $M = \{(C^i \mapsto C^{i+1})\}$, $i = 0 \ldots n-1$ is a path of an Elp model (i.e. an n-ary relation). The BJ-Measure of a path M can be evaluated if a BJ-Measure exists for each relation of the series $\{(C^i \mapsto C^{i+1})\}$. Indeed, a path M containing a relation $C^i \mapsto C^{i+1})$ such that $BJM(C^i \mapsto C^{i+1}) = 0\}$, $i < n-1$, means that there is no information flowing from the C^i class to the C^{i+1} class. No information can flow from the C^0 class to the C^n class: the BJ-Measure of the path M is then null. This leads to the following condition:

Definition 3 *The BJ-Measure of an n-ary relation $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$ exists if and only if, $\forall (C^i \mapsto C^{i+1}) \subseteq M$, $BJM(C^i \mapsto C^{i+1}) > 0$.*

The condition of the Definition 1 has been included in the BJ4T algorithm to produce only trees with timed sequential binary relations having a positive BJ-Measure. The BJ-Measure of a path M can then be defined as following:

Definition 4 *When it exists, the BJ-Measure of an n-ary relation $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$ is the sum of the BJ-Measure of each binary sequential relation $C^i \mapsto C^{i+1}$ of M.*

$$\begin{aligned} BJM(M) &= BJM(\{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}) \\ &= \textstyle\sum_{i=0,\ldots,n-1} BJM(C^i \mapsto C^{i+1}) \end{aligned} \quad (11)$$

The quantity $BJM(M)$ can then be interpreted as an estimation of the quantity of information that flows through the n-ary relation M. It is then possible to reason on this quantity.

Pruning a Chronicle Model

The Chapmann-Kolmogorov equation of the Markov chain theory provides the probability of a path $(i, i+1, \ldots, n)$ in the transition probability matrix of a given sequence ω (Le Goc & Bouché 2005): $p(i, i+1, \ldots, n) = \prod_{i=0,\ldots,n-1} p(C^{i+1}|C^i)$. This corresponds to the probability of a path $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$:

Definition 5 *The probability of an n-ary relation $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$ is the probability of the path $(i, i+1, \ldots, n)$ in the Markov chain corresponding to ω, and this probability is given by the Chapmann-Kolmogorov equation:*

$$p(M) = \textstyle\prod_{i=0,\ldots,n-1} p(C^{i+1}|C^i) \quad (12)$$

By definition, the quantity $P(M)$ decreases exponentially with the number n of relations in the path M when he quantity $BJM(M)$ increases monotonically. As (Smyth & Goodman 1992) with the J-Measure or (Van Rijsbergen 1979) with the F-measure, these two quantities can be combined to find a good tradeoff between the probability of a path M (i.e. its generality) and the quantity of information flowing through it (i.e. its quality). (Le Goc & Benayadi 2007) shows that this trade off is proportioned with the number n of binary sequential relation in the path M. We defined then the quantity $L(M) = n \times P(M) \times BJM(M)$ to represent this trade off. By construction, $\lim_{n\to 1}(L(M)) = L(C^0 \mapsto C^1) > 0$ and $\lim_{n\to\infty}(L(M)) \to 0$. Between these two limits, $L(M)$ is bounded by a convex function the maximum of which depends of the marginal probability of the discrete event classes that composes the n-ary relation ((Benayadi & Le Goc 2007)). This leads to the following heuristic for pruning a path $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$ when there is a sub-path M^1 of M that maximize the quantity L:

Definition 6 *Given a path $M = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-1}$, M can be decomposed into tree series $M^1 = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-k-1}$, $M^2 = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-k}$ and $M^3 = \{C^i \mapsto C^{i+1}\}_{i=0\ldots n-k+1}$, $k \geq 1$, so that:*
$L(M^1) \leq L(M^2) > L(M^3)$.

This definition is an heuristic because there is no guarantee that this maximum is global. But this local maximum corresponds to a kind of optimum between the generality and the quality of an n-ary relation. This heuristic is simple enough to be evaluated in a new algorithm called the "BJT4P" (BJT for Pruning) algorithm. This algorithm implements this heuristic to prune a tree provided by a new version of the BJT4T algorithm that implements to the condition of the Definition 1. The next section shows on an industrial application the results this BJ-Measure based heuristic provides.

Application to SACHEM

Sachem is the name of the very-large-scale knowledge-based system the Arcelor Mittal Steel group has developed at the end the 20th century to managed its production tools. Sachem is design to help the operators to monitor, diagnose and control production tools like the blast furnace, one of the most complex production process (Le Goc 2004). Sachem is also the generic core of other knowledge based systems that manage other kind of production tools like a galvanization bath (the Apache system, cf. (Le Goc & Benayadi 2007), (Le Goc & Bouché 2005)). The works presented in this paper concerns the design of the second generation of the

Sachem systems with the aim to propose a high level diagnosis service and to minimize the knowledge acquisition and modeling phase. The diagnosis function aims at producing a causal interpretation of the observed behavior so that adequate actions could be recommended to avoid potential problems.

With a Sachem system, the behavior is described with a flow of occurrences of phenomenon (i.e. a timed instance of a phenomenon). A phenomenon corresponds to the logical description of a type of physical or chemical transformation that can occur in a process. A phenomenon is represented with an object class characterized by a name, an identifier, a set of attributes and four dates: begin date, end date, begin detection date, end detection date. An occurrence of a phenomenon is then an instance of such a class where attributes and dates are valuated by the perception function of a Sachem system. A phenomenon occurrence is then created when a behavior corresponding to a particular phenomenon is recognized. The creation is therefore the affectation of values to the attributes and specifically the begin and begin detection date ones. A phenomenon occurrence is deleted when the end of a phenomenon is detected. So the end date and end detection date attributes are valuated. A behavior is a series of phenomenon occurrence ordered by their begin date. Such a series is considered as a sequence of discrete event classes where the classes are the observed phenomena.

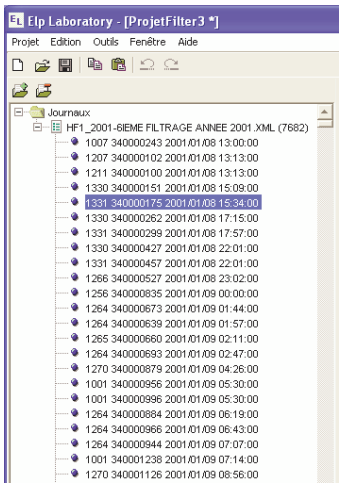


Figure 5: Beginning of the ω Sequence

The figure 5 presents the beginning of the ω sequence produced by Sachem at Fos-Sur-Mer (France) between the 08/01/2001 and the 31/12/2001. The whole sequence contains 7682 occurrence of 45 discrete event classes (i.e. phenomena). The associated Markov chain is made of $45 \times 45 = 2025$ states corresponding to a superposition of 2025 Poisson processes.

The application of the new version of the BJT4T algorithm produces the tree of the most probable timed binary sequential relations of the Figure 6) for the arbitrary chosen 1463 class. This class is concerned with the usage of the gas inside the blast furnace: an occurrence of the 1463 class means that the gas is not well used. This phenomenon is the results of a wrong management of the blast furnace. The algorithm was parameterized to build a tree of 5 classes depth and 20 classes width. Such a tree containing so $20^5 = 3200000$ nodes, it is handily very difficult to analyze.

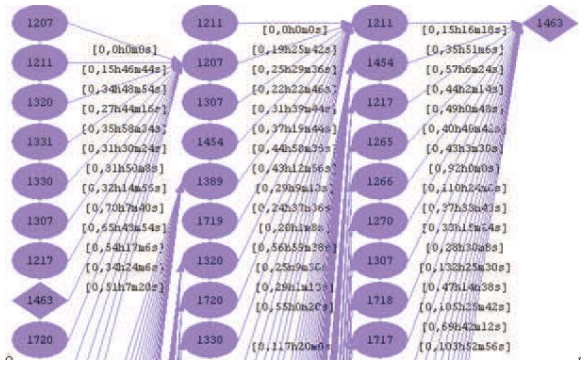


Figure 6: BJ4T tree for the 1463 Class

The Figure 7 shows the pruned tree of Figure 6 produced with the BJT4P algorithm described in the preceding section. This tree contains 195 nodes, that is to say a reduction factor of 100,000. To analyze the properties of the pruning algorithm, let us look for the pathes having an anticipating ratio greater than 50% with the BJT4S algorithm (cf. Figure 8). Such a path is called a "signature" because it is a good candidate to be a diagnosis rule.

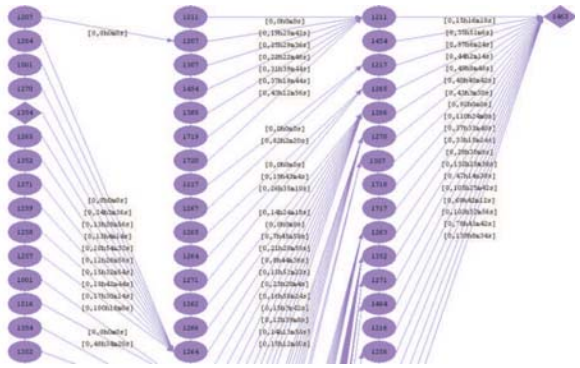


Figure 7: 1463 Class BJT4T Pruned Tree

The BJT4S algorithm produces the signatures of Figure 8 with the pruned tree of Figure 7. The anticipating and the cover ratio of the signatures are noted on the timed binary sequential relation (in the Figure 8, AR and CR mean respectively Anticipating Rate[1] and Cover Rate).

This result is very good: all the discovered signatures corresponds to the *a priori* knowledge about the causes

[1] as it is defined in (Le Goc & Bouché 2005), the Anticipating rate can be greater than 100%.

of a modification of the omega variable in a blast furnace (Le Goc 2006). Technically, such a result was almost impossible to obtain without a pruning principle, that is to say without taking into account the Information Theory in the Stochastic Approach. But more, if the pruning heuristic does not guarantee that all the signatures of a given tree are still contained in the corresponding pruned tree, this application shows that obviously, the signatures contained in the pruned tree have a strong meaning according to the *a priori* knowledge about the concerned phenomena. The same result is observed on the Apache system, a clone of Sachem design to manage galvanization bathes. This shows that the BJ-Measure is an interesting tool to help the mining of a sequence of alarms according to the Stochastic Approach.

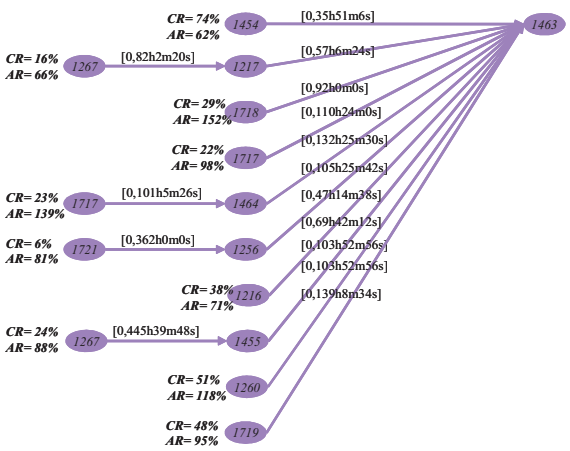


Figure 8: 1463 Class Signatures

Conclusion and Perspective

This paper proposes a new measure for discovering the chronicles models from a sequence of discrete event class occurrence. This measure is an adaptation of the J-Measure to ordered set of data. When combined with the Markov chain theory (i.e. the Stochastic Approach for discovering chronicle models), the BJ-Measure allows to envisage the principles of an unsupervised learning process.

The Stochastic Approach provides a global description of the sequence under the form of a probability transition matrix of a Markov chain from which the BJT4T algorithm deduced a tree of the most probable chronicle models. The BJ-Measure allows the definition of heuristics for pruning these trees with the aim of reducing the search space of potential diagnosis rules. Compared to the existing methods of the literature, our approach presents the following advantages:

- The Stochastic Approach allows to extract a set of potential diagnosis rules in only one reading of the sequence. This is a great advantage when the sequence is huge, that is to say where most of the proposed methods have computing problems.
- The discovered chronicle models have a strong semantic according to the dynamic process to diagnose because the n-ary relations are looked for a specific discrete event class.

The algorithm described in this paper is one of the tools developed in "ELP Laboratory", a Java environment for analyzing the discrete events sequences. Our currents works are concerned with the definition of heuristic for pruning a tree according to its width, the comparison of our approach with the other approach and the application the modeling to a complex manufacturing process.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. *Proceedings of the 11th International Conference on Data Engineering (ICDE95)* 3–14.

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* 207–216.

Benayadi, N., and Le Goc, M. 2007. Combining the j-measure and a markov chain to discover temporal knowledge. *Submitted to the The 13th International Conference on Knowledge Discovery and Data Mining, San Francisco, California, USA*.

Blachman, N. 1968. The amount of information that y gives about x. *Information Theory, IEEE Transactions* 14.

Blanchard, J.; Guillet, F.; Gras, R.; ; and Briand, H. 2004. Mesurer la qualité des règles et de leurs contraposées avec le taux informationnel tic. *Revue des Nouvelles Technologies de l'Information* E:287–298.

Blanchard, J.; Guillet, F.; Gras, R.; and Briand, H. 2005. Using information-theoretic measures to assess association rule interestingness. *5th IEEE International Conference on Data Mining (ICDM'05), IEEE Computer Society Press* 66–73.

Bouché, P.; Le Goc, M.; and Giambiasi, N. 2005. Modeling discrete event sequences for discovering diagnosis signatures. *Proceedings of the Summer Computer Simulation Conference (SCSC05) Philadelphia, USA*.

Brin, S.; Motwani, R.; and Silverstein, C. 1997. Beyond market baskets. generalizing association rules to correlations. *Proc. of ACM SIGMOD 97* 265–276.

Cauvin, S.; Cordier, M.-O.; Dousson, C.; Laborie, P.; Lévy, F.; Montmain, J.; Montmain, M.; Porcheron, M.; Servet, I.; and Travé, L. 1998. Monitoring and alarm interpretation in in-dustrial environments. *AI Communications ,IOS Press , 1998* 139–173.

Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 261283.

Cordier, M., and Dousson, C. 2000. Alarm driven monitoring based on chronicle. *Proceedings of SafeProcess, Budapest, Hun-gary* 286–291.

Dousson, C., and Duong, T. V. 1999. Discovering chronicles with numerical time constraints from alarm logs for

monitoring dynamic systems. *In D. Thomas, editor, Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)* 1:620–626.

Ghallab, M. 1996. On chronicles: Representation, on-line recognition and learning. *Proc. Principles of Knowledge Representation and Reasoning, Aiello, Doyle and Shapiro (Eds.) Morgan-Kauffman,* 597–606.

Gras, R.; Kuntz, P.; Couturier, R.; and Guillet, F. 2001. Une version entropique de lintensité dimplication pour les corpus volumineux. *Actes des journées Extraction et Gestion des Connaissances (EGC)* 6980.

Gras, R.; Couturier, R.; Blanchard, J.; Briand, H.; Kuntz, P.; and Peter, P. 2004. Quelques critères pour une mesure de qualité de règles dassociation. *Revue des Nouvelles Technologies de lInformation E-1 (2004), numéro spécial Mesures de qualité pour la fouille de données* 331.

Gras, R. 1996. *Implication statistique nouvelle méthode exploratoire de données*. France: La Pensée Sauvage Editions.

Hanks, S., and Dermott, D. M. 1994. Modeling a dynamic and uncertain world i: symbolic and probabilistic reasoning about change. *Artificial Intelligence* 1–55.

Hatonen, K.; Klemettinen, M.; Mannila, H.; Ronkainen, P.; and Toivonen, H. 1996a. Knowledge discovery from telecommunication network alarm databases. *In 12th International Conference on Data Engineering (ICDE '96). New Orleans, LA* 115–122.

Hatonen, K.; Klemettinen, M.; Mannila, H.; Ronkainen, P.; and Toivonen, H. 1996b. Tasa: Telecommunication alarm sequence analyzer, or how to enjoy faults in your network. *In IEEE Network Operations and Management Symposium (NOMS '96). Kyoto, Japan* 520–529.

Jaroszewicz, S., and Simovici, D. A. 2001. A general measure of rule interestingness. *In Proceedings of PKDD2001, Springer-Verlag* 253265.

Kodratoff, Y. 2000. Extraction de connaissances partir des données et des textes. *in Actes des journées sur la fouille dans les données par la méthode danalyse statistique implicative* 151165.

Le Goc, M., and Benayadi, N. 2007. A sequential j-measure to mine discrete event sequences. *Submitted to the 18th European Conference on Machine Learning (ECML) and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Warsaw, Poland.*

Le Goc, M., and Bouché, P. 2005. Analyse stochastique de séquences d'événements discrets pour la découverte de signatures. *Revue des Nouvelles Technologies de l'Information - Extraction et gestion des connaissances ISSN:1764-1667* 1:103–114.

Le Goc, M. 2004. Sachem, a real time intelligent diagnosis system based on the discrete event paradigm. *Simulation, The Society for Modeling and Simulation International Ed.* 80(11):591–617.

Le Goc, M. 2006. *Notion d'observation pour le diagnostic des processus dynamiques: Application Sachem et la dcouverte de connaissances temporelles*. HDR, Faculté des Sciences et Techniques de Saint Jérôme.

Loevinger, J. 1947. A systematic approach to the construction and evaluation of tests of ability. *Psychological Monographs 61* 4.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 259–289.

Mannila, H. 2002. Local and global methods in data mining: Basic techniques and open problems. *9th International Colloquium on Automata, Languages and Programming, Malaga, Spain,* 2380:57–68.

Sebag, M., and Schoenauer, M. 1988. Generation of rules with certainty and confidence factors from incomplete and incoherent learning bases. *in Proceedings of the European knowledge acquisition workshop EKAW88, Gesellschaft fr Mathematik und Datenverarbeitung mbH* 128.

Shannon, C., and Weaver, W. 1949. The mathematical theory of communication. *University of Illinois Press*.

Shore, J., and Johnson, R. 1980. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *Information Theory, IEEE Transactions* 26:26–37.

Smyth, P., and Goodman, R. M. 1992. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering 4* 301316.

Theil, H. 1970. On the estimation of relationships involving qualitative variables. *American Journal of Sociology* 103154.

Van Rijsbergen, C. J. 1979. *Information Retrieval*. London: Butterworth, , second edition.

HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis

Sriram Narasimhan[1] and Lee Brownston[2]

[1]University of California, Santa Cruz and [2]Perot Systems Government Services, Inc.
M/S 269-3, NASA Ames Research Center, Moffett Field, CA-94035
[1]sriram,[2]lbrownston@email.arc.nasa.gov

Abstract

In recent years several approaches have been proposed for model-based diagnosis of hybrid systems. These approaches deal with discrete or parametric faults, and perform consistency-based, stochastic or mixed reasoning. The major restriction that a diagnosis application designer faces is that each technique uses its own modeling paradigm and the reasoning algorithms implement a single strategy. Diagnosis application designers would like to have the flexibility of building models that are suitable for the task (*e.g.*, appropriate abstraction level, appropriate details, type of modeling paradigm used). They would also like to have the flexibility in choosing the strategies used in the diagnostic reasoning process.

We propose a general framework for stochastic and hybrid model-based diagnosis called Hybrid Diagnosis Engine (HyDE) that offers this flexibility to the diagnosis application designer. We list the key steps in stochastic and hybrid diagnosis and identify the kinds of models that are needed in those steps. The HyDE architecture supports the use of multiple modeling paradigms at the component and system level. Several alternative algorithms are available for the various steps in diagnostic reasoning. This approach is extensible, with support for the addition of new modeling paradigms as well as diagnostic reasoning algorithms for existing or new modeling paradigms. We discuss the current status of HyDE and its application in diagnosis of real and conceptual systems.

Introduction

The traditional approach to building a model-based diagnosis system is to select a diagnosis technology, build models targeted for the selected technology, and test the reasoning algorithms on the models using real or simulated data to refine the models and fine tune the performance of the algorithms. This approach works well when the diagnosis application designer is familiar with the diagnosis technology and the modeling paradigm it uses and has a good idea of how to design the models to achieve a specific goal, such as the diagnosis of a specific set of pre-selected faults. In many cases, however, the diagnosis problem is not so well defined and the diagnosis designer would like to have the flexibility to experiment with different kinds of models (*e.g.*, multiple paradigms and abstraction levels) as well as different strategies for diagnosis reasoning (*e.g.,* pure consistency-based, purely stochastic and different kinds of search algorithms).

In this paper we present the Hybrid Diagnostic Engine (HyDE), a general framework for stochastic and hybrid model-based diagnosis of discrete faults, that is, spontaneous changes in operating modes of components. HyDE combines ideas from consistency-based [Hamscher, *et al*., 1992; De Kleer and Williams, 1987] and stochastic [Hofbaur and Williams, 2002; Dearden and Clancy, 2002] approaches to model-based diagnosis using discrete [Williams and Nayak, 1996; Kurien and Nayak, 2000], continuous [Gertler, 1988; Mosterman and Biswas, 1999] and hybrid [Narasimhan and Biswas, 2003; Hofbaur and Williams, 2002; Dearden and Clancy, 2002] models to create a flexible and extensible architecture for stochastic and hybrid diagnosis. HyDE supports the use of multiple paradigms and is extensible to support new paradigms. HyDE offers the application designer a wide variety of options in selecting the diagnosis reasoning strategy. The key features of HyDE are:

- Diagnosis of multiple discrete faults.
- Support for hybrid models, including autonomous and commanded discrete switching.
- Support for stochastic models and stochastic reasoning.
- Capability for handling time delay in the propagation of fault effects.

Some preliminary work related to HyDE has been presented in [Narasimhan, *et al*., 2003; Narasimhan, *et al*., 2004].

The following sections first present the kinds of models used in HyDE, each of which includes a generic transition model and a modeling-paradigm-specific behavior model. Then we describe the kinds of faults HyDE can diagnose and how they are represented in the models. Next we discuss the HyDE reasoning architecture and how it supports the use of diverse algorithms, enabling various diagnosis strategies. We then identify the features currently implemented and finally describe some applications that use HyDE for diagnosis.

HyDE Models

HyDE models have two parts. The first, which is common to all supported modeling paradigms, describes the transition behavior of the system. The second is the behavior model, which is specific to each modeling paradigm.

The transition model describes the following elements of the system:

1. The set L of operating modes l_i of the system, $1 \leq i \leq |L|$, called *Locations*.
2. The set T of allowed *transitions* t_i, $1 \leq i \leq m$, between the locations of the system, where $t_i = l_j \rightarrow l_k$ indicates a transition from location l_j to l_k.

The behavior model specifies the behavior evolution and has three parts: propagation model, integration model and dependency model. Each kind of model part is used for a different step in the diagnosis reasoning process. The information in the propagation model allows the estimation of unknown variable values from known variable values. The dependency model captures information about the dependencies between variables, models and components. The integration model describes how the variables' values are propagated across time steps. Depending on the modeling paradigm used, the same model may serve all three roles or it may be necessary to specify each model independently. The integration model is necessary only if there are variables that have state, *i.e.*, if values at one time step depend on values at previous time steps. The dependency model is optional, since a fully-connected graph may be used as a trivial dependency model, but Candidate Generation can be optimized if a dependency model is specified. The behavior model is expressed as:

1. The set V of *variables* v_i in the system, $1 \leq i \leq |V|$.
2. The set of D *domains* d_i, $1 \leq i \leq |V|$, specifying the allowed values (data types) for the variables, where d_i represents the domain for variable v_i.
3. The set G of transition *guards* g_i, $1 \leq i \leq |T|$, representing the conditions for system transitions, where g_i is a condition predicate for transition t_i.
4. The *propagation model* PM specifies the behavior of the system within a time step[1] as relations over variables. This includes
    a. Global model $PM_g = R_g(V)$, where R_g is the set of relations constraining values of variables expressed in one of the supported modeling paradigms.
    b. Local models $PM(l_i) = R_{li}(V)$ for each $l_i \in L$, where R_{li} is the set of relations constraining values of variables expressed in one of the supported modeling paradigms.
5. The *integration model* IM specifies the evolution of values each variable across time steps. $IM(v_i) = R_t(v_i(t_k), \delta v_i(t_k), v_i(t_{k-1}), \delta v_i(t_{k-1}), \ldots, v_i(t_{k-n}), \delta v_i(t_{k-n}))$, where R_t is a relation, $v_i(t_k)$ is the instance of variable v_i at time t_k, $\delta v_i(t_k)$ is the instance of the derivative of v_i at time t_k.
6. The *dependency model* DM specifies dependencies among variables in V, relations R_g and R_{li} associated with the propagation model and relations R_t associated with the integration model.

For the sake of modularity, composability and hierarchy, HyDE supports the use of component models as well as system models. Component models specify the system model in terms of the transition and behavior models of the individual components and the interconnections among components. Diagnostic reasoning may be performed directly on component models in some cases, but in most cases the component models have to be composed to a higher-level system model before use. Users can choose to build system models directly. The system model is composed from the component model as follows:

- The transitional model for the system is a synchronous composition of the transitional models of the components and is derived as follows:
    - $SL = L_1 \times L_2 \ldots \times L_n$ where 1, 2 … n are the indices of the components making up the system. A system location sl_1 would be $(l_{1i}, l_{2j}, \ldots, l_{nk})$, where the first subscript indexes the component and the second subscript indexes a location in that component.
    - The system transitions T_s are derived as follows: For each transition for component c $l_{ca} \rightarrow l_{cb}$, corresponding transitions are created from every location in the system that includes l_{ca} in the label to the location that contains the label l_{cb}, with the remaining label being the same.
- The composition of the behavior model is a specialized algorithm that depends on the type of modeling paradigm used for the system model. The system behavior model is derived as follows:
    - $V_s = V_g \cup V_1 \cup V_2 \ldots \cup V_n$, where 1, 2 … n are the indices corresponding to the components making up the system and V_g represents the set of variables that do not belong to any component.
    - The transition guards remain the same for corresponding transitions.
    - For each location $l_s = \{l_{1i}\, l_{2j} \ldots l_{nk}\}$. $M_s = M_g \cup M_{connection} \cup M_{1i} \cup M_{2j} \ldots \cup M_{nk}$ where the union operation U depends on the modeling paradigm used for component and system models. $M_{connection}$ is the component connection model that specifies how components interact with each other. M represents propagation, integration and dependency models.

[1] Variables are assumed to take one or more values from their domains at each time step in the reasoning process described later.

HyDE also supports multiple behavior model paradigms at the component and system level. Component models may be specified in one paradigm and then transformed to another form before composition to system model form; or the component models may be transformed to a system model in one paradigm, which can then be transformed to a system model in another paradigm. Figure 1 illustrates the interaction between component and system models. The following convention is used in Figures 1 through 5.

Both the transformation and composition may be performed in either order when needed, rather than pre-compiling system models in all possible system locations.

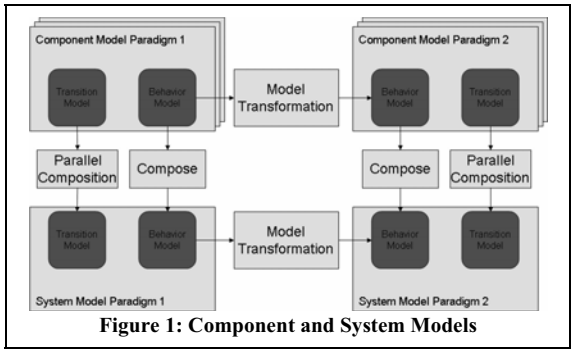


Figure 1: Component and System Models

HyDE Faults

HyDE can deal with discrete faults, that is, those that correspond to an abrupt change in the configuration of the system. HyDE models represent discrete faults as transitions without guards (called unguarded transitions). The absence of guards signifies that the occurrence of these transitions cannot be directly observed, and hence must be determined by reasoning about symptom observations. Examples of such faults are valve stuck open, motor stalled and circuit breaker tripped. Unexpected behavior not explicitly modeled can be handled as an unguarded transition to a unique "unknown" location that has no model associated with it, assuring that it will be consistent with all observations. It is also possible to model parametric faults (abrupt changes in the values of system parameters) if the new values for the parameters are known. For example, we can model resistive faults as an n% change in resistance for several pre-specified values of n. It is not possible to model the general case in which n is not known and has to be inferred by reasoning.

The use of unguarded transitions allows HyDE to infer the occurrence of any unobserved event, not just faults. For example, a transition that is based on a supervisory controller command may be represented as an unguarded transition if the issuance of the command is not observed.

HyDE Reasoning

HyDE reasoning is the maintenance of a set C of weighted candidates (c_i, w_i), $1 \leq i \leq k$. A candidate represents the hypothesized alternative trajectories of the system inferred from the transition and behavior models of the system, knowledge of the initial locations of all components and initial values of all variables, and the sensor observations reported to HyDE. The candidates' weights are a way of ranking them and depend on several factors, including prior probabilities of transitions and the degree of fit between model predictions and observations. Although weights are in the range [0, 1], weight is not a probability measure, specifically posterior probability.

Each candidate contains a possible trajectory of system behavior evolution represented in the form of a *hybrid state* (HS) history and transition history. The hybrid state is a snapshot of the entire system state at any single instant. It associates all components with their current locations and all variables with their current values. HS = (SL, VV) where $SL = \{(comp_i \rightarrow l_i) \mid 1 \leq i \leq n\}$, and $VV = \{(v_i \rightarrow value_i) \mid 1 \leq i \leq m\}$. Applications run HyDE at discrete time steps, typically but not necessarily when observations are available. Time steps need not be periodic. For each time step that HyDE reasons about, a candidate contains two hybrid states, one at the beginning of the time step and one at the end, as well as the set of transitions taken by the system between the previous and current time steps.

If HyDE has been run for a set of time steps $\{t_i \mid 1 \leq i \leq end\}$, a candidate will be of the form $\{(Trans_{\rightarrow ti}, hs_{ti-}, hs_{ti+}) \mid 1 \leq i \leq end\}$, where $Trans_{\rightarrow ti} = \{(comp_i, trans_i) \mid 1 \leq i \leq end\}$ is a set of ordered transitions (with associated component) believed to have been taken by the system between time steps t_{i-1} and t_i, hs_{ti-} indicates the hybrid state at the beginning of time step t_i, and hs_{ti+} indicates the hybrid state at the end of time step t_i. All components that do not have an explicit transition in $Trans_{\rightarrow t}$ are assumed to have taken an implicit special guarded transition called self-transition $trans_{li}$. This transition is from location l_i to location l_i. As a result, $Trans_{\rightarrow t}$ will include an entry for all components but may have more than one entry for some components if it is believed that more than one transition occurred for those components.

At time step 0 (or some user-specified start time step t_i) the candidate set is initialized with candidate(s) derived from the initial hybrid state of the system. If there is some uncertainty about the initial hybrid state, it is possible to sample from this uncertainty to create a set of initial candidates. In the worst case, when the initial hybrid state is not known, a set of candidates may be created by randomly sampling the locations of the components. $Trans_{\rightarrow 0}$ is set to contain the self transition for all

components. After initialization a candidate c_i would look like $\{(Trans_{\rightarrow 0}, hs_{0-}, hs_{?})\}$, where $hs_{?}$ indicates an unknown or not yet estimated hybrid state. The weights of the candidates may be set to 1; if prior probabilities are available for initial hybrid states, those values can be used as initial weights.

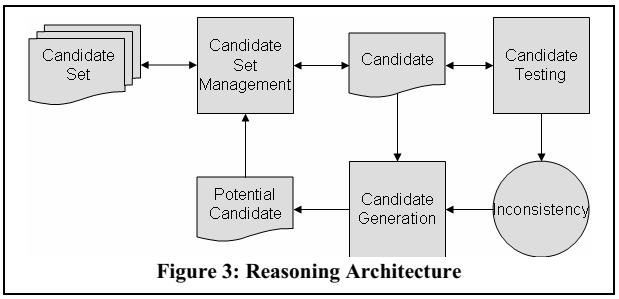


Figure 3: Reasoning Architecture

Once the initial candidate set has been created, HyDE's reasoning process uses the same sequence of operations for each time step. The reasoning process can be divided into three categories of operations, as illustrated in Figure 3:

1. Candidate Set Management maintains the candidate set, including pruning unlikely candidates and adding new candidates when necessary.
2. Candidate Testing deals with operations on a single candidate, including estimation of the hybrid state, updating the weight of candidate and reporting inconsistencies.
3. Candidate Generation creates candidate generators from inconsistencies reported by Candidate Testing and supplies the next-best potential (untested) candidate to Candidate Set Management when requested.

In the next three sections we will discuss each of these categories in more detail.

Candidate Set Management

There are four Candidate Set Management operations, shown in sequential order in Figure 4.

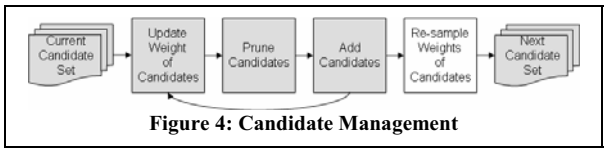


Figure 4: Candidate Management

Updating weight of all candidates. The Candidate Testing operations are called to update the weight of each candidate in the candidate set. When candidates are tested, additional candidates may be spawned (described in the candidate-testing section). Based on user preferences, these candidates may be added to the candidate set for testing; incorporated into a candidate generator to be re-generated when requested (for later testing); or completely ignored.

Prune candidates. After the candidate weights have been updated, candidates with weights below a user-specified threshold t_w are pruned from the candidate set: $C \leftarrow \{ c_i \mid c_i \in C \ \& \ w_i >= t_w \}$.

Add candidates. Additional candidates may be needed to fill the candidate set to a user-specified minimum count. To re-fill the candidate set, a new potential candidate is requested from the bank of candidate generators created as a by-product of the candidate-testing operations. The candidate generators are responsible for selecting the next best candidate based a user-specified ranking function. The potential candidate then goes through Candidate Testing and may be pruned if its updated weight is not high enough. More potential candidates are requested until the candidate set has the requisite number of candidates.

Re-sample weights. As an optional step, candidate weights may be normalized by re-sampling from the distribution of weights. For the candidate set $C = \{(c_i, w_i) \mid 1 \le i \le k\}$, the sum $W = \sum w_i$, $1 \le i \le k$ is determined and the candidate set is cleared $C = \{\}$. A random real number x is sampled between 0 and W. If the value of x is between $\sum w_i$, $1 \le i \le j-1$ and $\sum w_i$, $1 \le i \le j$, candidate $(c_j, 1)$ is added to the candidate set. This process is repeated until the candidate set has the requisite number of candidates (usually k).

Candidate Testing

At each time step t_i, each candidate c_j is tested to find the enabled transitions $Trans_{\rightarrow ti}$ taken between the previous time step t_{i-1} and the current time step t_i; to estimate the hybrid state at the beginning of the time step hs_{t1-}; to compose the system propagation model PM_{ti}; to estimate the hybrid state at end of time step hs_{t1+}; update the weight w_j of the candidate; and to report any inconsistencies I_{ij}. These steps are illustrated in Figure 2.

Find enabled transitions. First collect the set of all possible transitions out of the system location associated with hs_{ti-1}. The user has the option of including unguarded transitions in this list. The guards on the selected transitions are evaluated to compute a weight indicating the

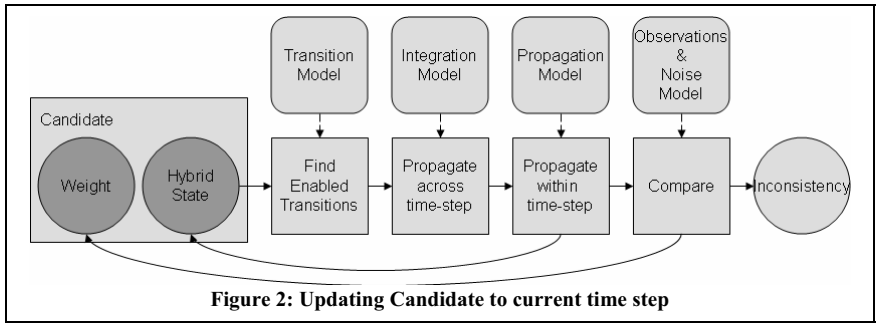


Figure 2: Updating Candidate to current time step

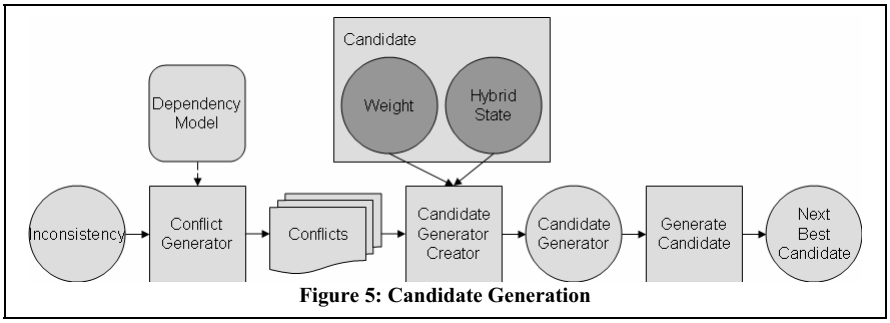


Figure 5: Candidate Generation

likelihood of the guard being true. The user has to specify a policy for handling enabled transitions. The options are to consider only the most likely transition with weight > 0.5, choosing the self transition if no such transition exists; sampling from the distribution of enabled transitions; selecting all transitions with weight > 0.5; or selecting all enabled transitions irrespective of weight.

Estimate beginning hybrid state. The estimation of the hybrid state at the beginning of the time step t_i involves two operations.

1. Estimate the new system location: First a new $Trans_{\rightarrow ti}$ is created for each enabled transition in the previous step. $Trans_{\rightarrow ti}$ is applied to $SL(hs_{ti-1})$ to determine $SL(hs_{ti-})$. There will be as many new $SL(hs_{ti-})$ as there are enabled transitions. As mentioned earlier, all but the most likely one are reported to Candidate Set Management, which decides what will be done with them.
2. Estimate the new variable values: The integration model IM_{ti} is applied to $VV(hs_{ti-1})$ to get $VV(hs_{ti-})$. Additionally, values for any input variables are set to be their sensed values, if reported, at t_i.

Load the propagation model. The possibly-new propagation model of the system corresponding to $SL(hs_{ti-})$ has to be loaded. If $SL(hs_{ti-})$ has been reached earlier in the reasoning process, its propagation model PM_{ti} has already been cached and can be loaded directly. If $SL(hs_{ti-})$ has not been reached, PM_{ti} has to be composed from the model associated with $SL(hs_{ti-})$, as described in the modeling section. The composed model is cached for later re-use.

Estimate end hybrid state. The next step is to estimate hs_{ti+} using PM_{ti}. Assuming that all transitions occur between successive time steps, we set $SL(hs_{ti+}) = SL(hs_{ti-})$. $VV(hs_{ti+})$ is estimated by initializing PM_{ti} with $VV(hs_{ti-})$ and executing PM_{ti} by running a simulation or propagation algorithm. If the execution fails, an Inconsistency I_{ti} is reported to the Candidate Generation system. This inconsistency identifies the relation $r_{inconsistent}$ from PM_{ti} that did not hold.

Compare against observations. If output variables have been reported, HyDE compares these observed values Y_n to their predicted values $\hat{Y}_n$ in hs_{ti+}. The comparison step for n output variables is expressed as: $R_n = CF(\hat{Y}_n, Y_n, \varepsilon_n)$, where CF is a comparison function, R_n is a vector of n residual values in [0, 1] indicating the degree of fit and ε_n are the user-defined noise models for the n sensors associated with the output variables.

Update weight of candidate. An overall degree of fit $r_{overall}$ is also computed from R_n. The user may select this to be either the minimum weight in R_n, the arithmetic mean of R_n or some custom function over $(\hat{Y}_n, Y_n, \varepsilon_n)$. The weight of the candidate is updated by multiplying by $r_{overall}$.

Reporting inconsistency. If $r_{overall}$ is less than a user-specified threshold, an Inconsistency I_{ti} is reported that contains all output variables whose degree of fit $r_i \in R_n$ is less than the same user-specified threshold.

Candidate Generation

The Candidate Generation system is responsible for creating a new potential candidate when requested by the Candidate Set Management system. Figure 5 illustrates the steps in Candidate Generation. The Candidate Generation system consists of a set of candidate generators. When a candidate is requested, each candidate generator reports its next best candidate, where "best" is a user-customizable criterion, and the best among these best candidates is selected and returned to the Candidate Set Management system.

Candidate generators come in two flavors. The first kind, based on a candidate, always reports that candidate until the candidate is selected for use. These kinds of candidate generators handle the situation in which Candidate Testing results in the spawning of new candidates and the user specifies that these candidates go into the set of candidate generators. In such a case each newly-spawned candidate becomes the basis for a candidate generator.

The second kind of candidate generator is based on *conflicts*. A conflict is a list of timed transitions which, if taken jointly by the system, result in an inconsistency. When the propagation or comparison step generates inconsistencies, conflicts are generated using the dependency model and become part of a new candidate generator. The next section describes how a conflict is created from inconsistencies and the section following describes how the conflict-based candidate generator works.

Conflict generation. When an inconsistency I_{ti} is reported at time t_i from the propagation or comparison step, HyDE creates (or retrieves if already created) the dependency model DM_{ti} corresponding to $SL(hs_{ti})$. For each component encountered, the transitions for that component from

$Trans_{\rightarrow ti}$ are added to the conflict set that has time stamp t_i. The same process is repeated in the dependency model DM_{ti-1} for the previous time step t_{i-1}. However, the starting points for the back traversals in the dependency model are all the variables that depend on I_{ti}. This adds transitions with time stamp t_{i-1} to the conflict set. The same procedure is repeated for n previous time steps, where n is a user-supplied parameter. The user may explicitly specify the amount of time to backtrack and the maximum number of guarded transitions to consider. When the inconsistency contains two or more variables, a conflict is created for each variable. A conflict-based candidate generator is then created containing an initial hybrid state hs_{ti-n} at time step t_{i-n}, an initial weight equal to the weight of the candidate that generated the inconsistency, and the sets of conflicts generated from the inconsistencies.

Generating a candidate from a conflict-based candidate generator. When requested for a candidate, the candidate generator generates a set $Trans_{unguarded}$ of timed unguarded transitions $\{trans_{ti} \mid 1 \le i \le k\}$ that resolve all the conflicts in the candidate generator. A conflict is resolved by a transition that is a sibling (same source location, different destination location) of at least one transition in the conflict. A set of transitions resolves a set of conflicts if, for each conflict, there is a transition in the set of transitions that resolves the conflict. The Candidate Generator saves its state so that the next time it is asked it can generate the next optimum candidate. The transitions are chosen so as to optimize some user-defined criteria. For example, the user may be interested in transitions with the maximum prior probability or the set of transitions with minimum size. Conflict resolution may use a two-step hitting-set-based solution (generate a hitting set and resolve the hitting set) or conflict-directed search [Williams and Ragno, 2003] to generate the transitions directly. A candidate is then created with initial hybrid state hs_{ti-1}, and pre-loaded $Trans_{\rightarrow t}$ for each t in $\{t_{i-n}, \ldots, t_i\}$. Each entry $trans_{tk}$ in $Trans_{unguarded}$ is added to $Trans_{\rightarrow tk}$.

HyDE Implementation Status

The HyDE reasoning engine is implemented in C++. Complete diagnosis reasoning can be performed although not all of the earlier-discussed capabilities have been implemented yet. It passes an extensive and demanding test suite on Windows, Solaris, Linux and VxWorks platforms. A graphical modeling environment is available using the GME open-source tool [Ledeczi, *et al*., 2001]. The same environment can also be used to set initial state and configuration parameters. Observations can be reported to HyDE either through streams (file or otherwise) or an API allowing integration with a real-time system.

HyDE model variables can have Boolean, Enumeration, Interval or Real domains. For Enumeration domains, the domain's values must be specified. The within-component transition model is specified as a finite-state automaton in which locations are the states and the transition guards may be specified either as commands, some predicate over model variables, or a combination of the two. The propagation model is specified as constraint predicates over model variables. Constraints may be Boolean expressions if the variables are Boolean; algebraic and ordinary differential equations for interval- and real-valued variables, and equality or inequality for all variables. If all variables are interval or real, the user can configure HyDE to solve the constraints symbolically to transform them into state space equations or a Kalman Filter when used for propagation. Currently the Euler integration technique is the only integration model. The dependency model can either be real-time justification (truth maintenance) mechanism or a dependency graph that is derived from the constraints.

HyDE currently supports only consistency-based Candidate Set Management. Candidates are added to the candidate set only if consistent and deleted only if inconsistent. The maximum number of candidates in the candidate set is configurable. In Candidate Testing, only identification of the first enabled transition is supported. Comparison may use an equality check or thresholding to make a binary determination of consistency. Candidate Generation uses best-first search with a configurable preference for what is considered best. The user can set several parameters, including maximum candidate size, minimum candidate weight, and time of a candidate's last-hypothesized fault transition to guide the search.

HyDE Applications

HyDE has been and is being used on several projects. These projects exercise very different capabilities of HyDE, affirming the need for such a general framework. The following three projects have successfully used HyDE in demonstrations using real-time telemetry streaming from the system being diagnosed.

1. **The Drilling Automation for Mars Environment (DAME)** project, led by NASA Ames Research Center, is aimed at developing a lightweight, low-power drill prototype that can be mounted on a Mars Lander and drill several meters below the Mars surface for conducting geology and astrobiology research. Three kinds of diagnosis technologies were used on this project, HyDE for

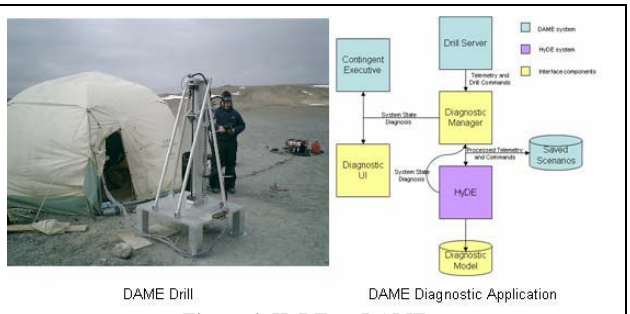


Figure 6: HyDE on DAME

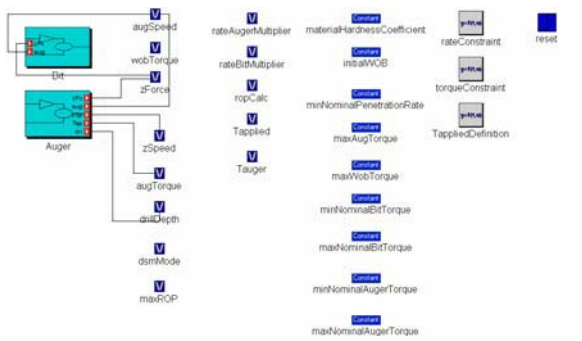


Figure 7: Top Level HyDE model of DAME

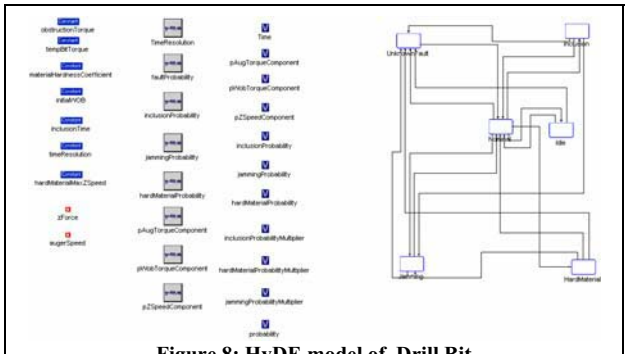


Figure 8: HyDE model of Drill Bit

model-based diagnosis, a rule-based diagnosis system, and a neural-network diagnosis system. The drill and the diagnostic application using HyDE are illustrated in Figure 6. HyDE was to model only the two major components of the drill, the bit and the auger (top level model shown in Figure 7). Nevertheless the model was complex, requiring 73 variables and 162 differential and algebraic constraints. The model had nominal (nominal, idle), faulty (jamming, inclusion unknownFault), and unobserved (HardMaterial) locations. The bit model is shown in Figure 8.

There were four rounds of testing over a period of two years. In 2005, laboratory experiments were run at Honeybee Robotics, the company that constructed the drill. Later in 2005, field tests were performed at Houghton Crater on Devon Island, in the Canadian arctic, chosen to approximate the Martian terrain and climate. Based on the results, laboratory tests were performed at NASA Ames Research Center in 2006 to improve the models for better diagnosis. Finally, there was a second field test at the Houghton Crater. [Balaban, *et al.*, 2007] summarize HyDE's final performance:

> All of the modeled fault modes were encountered in the field; some, such as choking, binding, and hard material, numerous times. The model-based diagnostic system was able to successfully identify the faults in roughly 85% of the cases. The rate of false positive diagnoses was approximately 5%.

2. **The Advanced Diagnostic and Prognostic Test-bed (ADAPT)** was developed at NASA Ames Research Center with the goal to test, measure, evaluate, and mature diagnostic and prognostic health-management technologies. The test-bed hardware for generates, stores, distributes, and monitors electrical power. The initial test-bed configuration is functionally representative of an exploration vehicle's Electrical Power System. HyDE was used to build an 86-component enumeration constraint model of the test-bed components. The ADAPT is in stark contrast to the DAME model in that it contains a lot of components but very few variables and constraints in the components. Diagnosis was purely consistency-based. In addition, modeling used a special feature of HyDE that allows observations to be made input variables, enabling the modelers to build consistency models instead of predictive models. HyDE passed all the acceptance tests, which included a set of pre-defined fault scenarios and acceptance bounds on the time to diagnosis. Additional tests were run on other fault scenarios (not included in acceptance tests) and HyDE was able to diagnose all but one of these scenarios (resulting from inadequate information in the model). More details of HyDE on ADAPT are presented in [Scott Poll, *et. al.,* 2007].

3. **The Autonomous Lander Demonstrator project (ALDER)** demonstrated autonomy capabilities relevant to a small spacecraft mission on traditional flight hardware integrated with traditional flight software. Diagnosis systems, including HyDE, were ported to VxWorks and executed on the Aitech S950 processor, interoperating with autonomy technologies for planning, diagnosis, computer vision and adaptive control. A HyDE component model of a simple propulsion system (tanks, valves, regulators, attitude control system and main engine) detected common faults such as stuck valves and regulator failures.

Several other projects are using HyDE for offline diagnosis using simulated data. The International Space Station (ISS) Electrical Power System (EPS) recovery procedure automation project at NASA Ames Research Center uses HyDE to supply current hybrid state to an execution system that will attempt to partially automate recovery procedures based on this information. The Aircraft Landing Gear Diagnosis project at NASA Langley Research Center is using HyDE to model the landing gear and wheels of an aircraft in an effort to diagnose faults during landing. The Spacecraft Engine Diagnosis project at NASA Marshall Space Flight Center uses HyDE to model components of the J2X engine, which is expected to power the upper stage of the NASA Crew Launch Vehicle (CLV). HyDE was also integrated with the CLARAty (Coupled Layer Architecture for Robotic Autonomy) architecture at NASA Jet Propulsion Laboratories.

Conclusions and Future Work

We presented the Hybrid Diagnosis Engine (HyDE), a general framework for stochastic and hybrid model-based diagnosis. HyDE supports multiple modeling paradigms and multiple strategies for the steps in the diagnosis reasoning. HyDE is extensible by adding new modeling

paradigms and user-defined algorithms for diagnosis reasoning steps. HyDE is being used on several projects that span the spectrum from consistency-based reasoning to stochastic reasoning and discrete models to hybrid models.

HyDE is still a work in progress and we expect to keep adding to the capabilities to HyDE through addition of new modeling paradigms and algorithms. We have identified several major areas for future work. We would like provide mechanisms for validation and verification (V&V) of models and algorithms, which would be prerequisite for deployment on real systems. We would like to add support for parametric faults. This would require support for modeling of parameters representing such faults, additional algorithms for parameter estimation and modifications to existing algorithms for detection and isolation of such faults. We would also like to use the existing models and algorithms for suggesting recovery sequences. Livingstone 2 [Kurien and Nayak, 2000] demonstrated how this can be done for finite-domain models. We would like to extend this to HyDE models and algorithms.

Acknowledgements. We thank the following people for providing us with information and pictures on the use of HyDE on their projects: DAME: Howard Cannon (NASA Ames Research Center), Edward Balaban (Perot Systems Government Services, Inc.); ADAPT: Scott Poll (NASA Ames Research Center), Adam Sweet (Perot Systems Government Services, Inc.); and ALDER: Jeremy Frank (NASA Ames Research Center), Edward Balaban (Perot Systems Government Services, Inc.). We also acknowledge the help of the following people in supplying information on the use of HyDE: Peter Robinson (Science Applications International Corp.), Vandi Verma (Perot Systems Government Services, Inc.), Cuong C. Quach and Sixto Vazquez (NASA Langley Research Center), Jon Patterson (NASA Marshall Space Flight Center), Bradley Burchett (Rose-Hulman University), and Thomas Slack (University of Memphis).

References

[Hamscher, *et al*., 1992] Walter Hamscher, Luca Console, and Johan De Kleer. *Readings in Model-based Diagnosis.* San Mateo, CA: Morgan Kaufmann, 1992.

[De Kleer and Williams, 1987] Johan De Kleer and Brian C. Williams. *Diagnosing multiple faults,* Artificial Intelligence, 32(1):97–130, 1987.

[Hofbaur and Williams, 2002] Michael Hofbaur and Brian Williams. *Mode Estimation of Probabilistic Hybrid Systems*, in Proc. 5th International Workshop on Hybrid Systems: Computation and Control (HSCC '02), Stanford, CA, USA, pp. 253-266, 2002.

[Dearden and Clancy, 2002] Richard Dearden and Dan Clancy. *Particle Filters for Real-time Fault Detection in Planetary Rovers*, in Proc. 13th International Workshop on Principles of Diagnosis (DX '02), Semmering, Austria, pp. 1-6, 2002.

[Williams and Nayak, 1996] Brian Williams and Pandu Nayak. *A model-based approach to reactive self-configuring systems*, in AAAI, pp. 971–978, (1996).

[Kurien and Nayak, 2000] James Kurien and Pandu Nayak. *Back to the Future with Consistency-based Trajectory Tracking*, AAA/IAAI 2000, pp370-377.

[Gertler, 1988] Janos Gertler. *Fault Detection and Diagnosis in Engineering Systems.* New York: Marcel Dekker, 1988.

[Mosterman and Biswas, 1999] Pieter J. Mosterman and Gautam Biswas. *Diagnosis of continuous valued systems in transient operating regions.* IEEE Transactions on Systems, Man, and Cybernetics, 1(6):554–565, 1999.

[Narasimhan and Biswas, 2003] Sriram Narasimhan and Gautam Biswas. *Model-based Diagnosis of Hybrid Systems.* Eighteenth Intl. Joint Conf. on Artificial Intelligence, Acapulco, Mexico, Aug., 2003.

[Narasimhan, *et al*., 2003] Sriram Narasimhan, Lee Brownston, and Daniel Burrows. *Explanation constraint programming for model-based diagnosis of engineered systems*, Aerospace Conference, 2004. Proceedings. 2004 IEEE, Vol.5, Iss., 6-13 March 2004 Pages: 3495- 3501 Vol.5.

[Narasimhan, *et al*., 2004] Sriram Narasimhan, Richard Dearden, and Emmanuel Benazera. *Combining Particle Filters and Consistency-based Approaches for Monitoring and Diagnosis of Stochastic Hybrid Systems*, 15th International Workshop on Principles of Diagnosis (DX04), Carcassonne, France, June 2004.

[Ledeczi, *et al*., 2001] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P.: *The Generic Modeling Environment*, Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.

[Williams and Ragno, 2003] Brian C. Williams, and Robert Ragno. *Conflict-directed A* and its Role in Model-based Embedded Systems*, Special Issue on Theory and Applications of Satisfiability Testing, Journal of Discrete Applied Math, January 2003.

[Balaban, *et al.*, 2007] Edward Balaban, Howard Cannon, Sriram Narasimhan, and Lee Brownston. *Model-Based Fault Detection and Diagnosis System for NASA Mars Subsurface Drill Prototype*, IEEE Aerospace Conference, Big Sky, Montana, March 3-10, 2007.

[Scott Poll, *et. al.,* 2007] Scott Poll, et. al. *Evaluation, Selection, and Application of Model-Based Diagnostic Tools and Approaches*, AIAA Infotech@Aerospace 2007, Rohnert Park, CA, May 7-10.

Symbolic Factorization of Propagation Delays out of Diagnostic System Models*

Jurryt Pietersma and **Arjan J.C. van Gemund**
Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {j.pietersma,a.j.c.vangemund}@tudelft.nl

Abstract

Many real-world systems exhibit time-dependent (dynamic) behavior such as time-dependent health, propagation time delays, and state. This behavior cannot be diagnosed by traditional algorithms that do not reason about time. Many different approaches have been proposed to include time in model-based diagnosis, which are essentially based on specific modeling techniques and associated reasoning algorithms. In this paper we propose an alternative approach, which addresses the problem of reasoning about time at the modeling level, rather than at the algorithm level. Central to our approach is the symbolic factorization of a time-dependent model into a time-independent part that can be solved by traditional algorithms, and a time-dependent part of which the solution is trivial. The approach only requires an off-line algorithm to factorize the model and does not require a new algorithm to diagnose dynamic systems during run-time. The modeling technique is based on modeling dynamic behavior through propagation delays, which also allows for modeling time-dependent health and state. We show that for systems whose dynamic behavior can be modeled in terms of propagation delays, our symbolic approach at the model level outperforms the essentially numeric approach at the algorithm level.

Introduction

Fault diagnosis of systems that exhibit time-dependent behavior such as copiers, spacecraft, cars, and wafer scanners, is a difficult problem and computationally hard. The theory of Model-Based Diagnosis (MBD) as initially proposed in (Reiter 1987) and in (de Kleer & Williams 1987) with the General Diagnostic Engine (GDE) does not consider time. When reasoning over time merely involves taking into account multiple observations in time, traditional techniques can be used. Essentially, for each time step the observation sample is combined with an instance of the system model, the conjunction of all combinations over time being solved by a traditional algorithm. The approach is essentially *numeric* in the sense that the time dimension is numerically expanded in terms of the problem that is input to the reasoning algorithm.

*This work has been carried out as part of the TANGRAM project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under grant TSIT2026.

This approach is also applicable to systems with time-dependent (intermittent) health. To illustrate this, we consider a valve as depicted in Figure 1. We model the valve as a system with an incoming and outgoing flow, denoted i and o respectively. For a healthy valve, the valve control

Figure 1: Valve.

variable, c determines o. A $true$ control variable implies an open valve for which o is equal to i, and a $false$ control variable implies a closed valve for which the outgoing flow is zero, i.e., $o = false$. For an unhealthy valve the healthy behavior is simply negated. Let h be the valve health, the valve model then becomes,

$$h \quad \Leftrightarrow \quad ((c \Rightarrow (o \Leftrightarrow i)) \wedge (\neg c \Leftrightarrow \neg o))$$

Let only c and o be observable. If at time t_1 the observations are $c(t_1) = false$ (closed valve) and $o(t_1) = true$ (flow out) then the diagnosis becomes $h(t_1) = false$ (leaky valve). Let the second set of observations be $c(t_2) = false$ and $o(t_2) = false$, indicating that the leak has stopped. If we assume $h(t_1) = h(t_2)$, i.e., a non-dynamic system, our model is no longer consistent as a broken valve at t_1 cannot explain the nominal behavior at t_2. Hence, we need to model a time-dependent health $h(t)$, which leads to a diagnosis $h(t_1) = false$ and $h(t_2) = true$.

When reasoning over time also involves taking into account other dynamic system behavior such as propagation delays and state (inherently a manifestation of time delay), a similar approach can be adopted as long as the delays are modeled in terms of discrete time samples. Many other approaches to modeling dynamic systems have been proposed (discussed later on) which are essentially based on the above numeric process in dealing with time. Essential to models of dynamic systems is that, apart from each variable (health, internal, or observable) becoming time-dependent, there may exist propagation time delays between components and the system may exhibit state. Examples of faults are time-outs of a system response, faults with delayed symptoms or with symptoms spread across time, intermittent faults, and faults that affect state behavior.

In this paper we propose an alternative approach, which addresses the problem of reasoning about time at the modeling level, rather than at the algorithm level. The approach is based on the *symbolic* factorization of a model that includes propagation time delays into a time-independent part that can be solved by any traditional algorithm that does not consider time, and a small, time-dependent part of which the solution is trivial.

The advantages of our approach are as follows. Unlike the above mentioned numeric approaches, this approach allows for the use of arbitrarily *real*-valued propagation delays within a model. Furthermore, our modeling-level, symbolic process significantly reduces diagnosis complexity compared to an algorithm-level, numeric approach. In the numeric approach with a K-step time window the diagnosis search time increases exponentially with K. In our approach the increase is determined by the number of connections between components. Let L denote the maximum number of connections to one component, then, for a worst case scenario, the diagnosis search time increases exponentially only with $K - \frac{K-1}{L}$ as will be shown later on. For engineered systems L is typically a small number (< 10) and hence $K - \frac{K-1}{L} \ll K$. Finally, the approach provides *transparency* as the problem of diagnosing dynamic systems is (symbolically) reduced to a traditional diagnosis problem, without the need for additional, time-specific, algorithmic approaches. To illustrate the symbolic technique and its reduction in computational complexity, we compare it with a simple, numeric approach for two toy problems (Polycell, a combinatoric example, and SR Latch, a state example), and present a performance evaluation based on the ISCAS-85 benchmark.

The remainder of this paper is organized as follows. In the first section we discuss related work. In the second section we present the concepts and notations for consistency based diagnosis, and also present our formal extension for dynamic systems. Next, we present two example models, which we use throughout the paper to demonstrate our methods. In the follow-on sections we explain the numeric expansion and the symbolic factorization method. Next, we present implementation, performance evaluation, and compare and discuss both methods. The last section states our conclusions.

Related Work

The importance of diagnosis of dynamic systems is evident from the fact that other approaches in the DX community have firmly included time in their approaches. Examples of these are timed failure propagation graphs (Abdelwahed, Karsai, & Biswas 2005), discrete event systems (Grastien, Cordier, & Largout 2005), and analytical redundancy of continuous, dynamic systems (Armengol *et al.* 2001). A full discussion of these approaches is beyond the scope of this paper in which we focus on including time in a GDE-like approach.

In the past, there has been a considerable amount of research on MBD of dynamic systems. MIMIC (Dvorak & Kuipers 1989) diagnoses continuous-variable dynamic systems with fault models derived from dynamic qualitative models. Diagnosis is performed at system sample rate under the assumption that faults occur one-at-a-time. In contrast to GDE, it does not use dependency tracing and makes use of explicit faults that may only occur in limited multiples. XDE (Hamscher 1991) is an extension to GDE that uses models with temporal abstractions created by the modeler, such as cycles, frequency, and change, to diagnose time-dependent, digital circuits. TEXSYS (Glass, Erickson, & Swanson 1991), diagnoses a prototype thermal control system by using qualitative and temporal reasoning over a combination of consistency and classification models.

Our approach does no require the user to model any artificial temporal abstractions and allows him to model systems from any domain with real-valued propagation delays. Furthermore we take benefit of an existing MBD implementation, LYDIA (Pietersma, Feldman, & van Gemund 2006) that also allows modeling of behavioral modes (de Kleer & Williams 1992), intermittent faults, and diagnoses multiple faults with a state-of-the-art, first-best, Conflict-directed A* search algorithm (Williams & Ragno 2004). Effort only has been made in the implementation of simple model conversion algorithms in front-end tooling.

Reasoning over time intervals also provides a powerful approach to the diagnosis of dynamic systems. The theory on time interval reasoning as proposed in (Williams 1990) was implemented for diagnosis in (Dague, Jehl, & Taillibert 1990) for continuous systems and in (Guckenbiehl & Schafer-Richter 1990) in a GDE-like fashion. We will show that our approach allows intervals to be constructed with delays and also accommodates for limited accuracy in the timing of observations.

More recent MBD implementations, Livingstone (Williams & Nayak 1999) and its successor Livingstone II (Kurien & Nayak 2000) employ a state approach in which the "next" operator allows modeling of time-dependent behavior. Livingstone implements time dependencies with state transitions and requires all transitions to be modeled synchronously. The combinatorial state explosion that is inherent to this approach is limited by this synchrony. Livingstone II further improves on this by regenerating past hypotheses based on saved history.

In our approach, state is easily modeled with recurrent (delay) relations, but does not require synchrony as arbitrary propagation delays are allowed. We believe this to be a more natural way of modeling multi-disciplinary systems. With factorization, the combinatorial explosion is curbed by taking advantage of the system's structure during compile time.

Preliminaries

We use the following notations and theory for our timed MBD. First, we present some basic terminology.

Definition 1 (System)**.** A diagnostic problem P is the ordered triple $P = \langle M, C, O \rangle$, where M is the system model, i.e., a set of propositional sentences describing the behavior of the system, C is a set of components, contained in the system, and O is an observation over some set of variables in M.

In this approach for each component $c \in C$ there is a corresponding propositional variable h_c representing its health

mode. We will call these variables h_c *health variables* and every instantiation of $\bigwedge_{c\in C} h_c$ a *health vector*.

Definition 2 (Diagnosis)**.** A *diagnosis* for the system $P = \langle M, C, O\rangle$ is a set $D \subseteq C$ such that $M \wedge O \wedge \left[\bigwedge_{c\in D} \neg h_c\right] \wedge \left[\bigwedge_{c\in(C\setminus D)} h_c\right] \not\models \bot$.

From Definition 1 and 2 it is visible that a diagnosis algorithm should use an entailment mechanism that finds the set of diagnoses D that are consistent with $M \wedge O$, i.e., D that can explain $M \wedge O$.

The system model M describes the system behavior in terms of relations between variables in a Finite Integer (FI) domain.

Definition 3 (FI Literal)**.** A FI variable $v_i \in V$ takes a value from a finite domain, which is an integer set S_i. A positive FI literal l_i^+ is a Boolean function $l_i^+ \equiv (v_i = s)$, where $v_i \in V, s \in S_i$. A negative FI literal l_i^- is a Boolean function $l_i^- \equiv (v_i \neq s)$, where $v_i \in V, s \in S_i$. If not specified, a literal l_i can be either positive or negative.

Note that all models in this paper have a Boolean domain, $S_i = \{False, True\}$, but that the proposed methods work for any FI domain.
The relations are expressed in Well-Formed Formula (Wff).

Definition 4 (FI Propositional Wff)**.** A FI propositional Wff is a formula over the FI literals $l_1, l_2, \ldots, l_n$, and the standard Boolean connectives $\neg, \Leftrightarrow, \Rightarrow, \wedge, \vee$.

Similarly, we define a time-dependent theory for MBD in which the system behavior, (health) variables and literals are a function of continuous time t, $t \in \mathbb{R}$. We define a *time-dependent* diagnostic problem $P(t)$, $P(t) = \langle M(t), C, O(t)\rangle$. Likewise we define time-dependent health variables, $h_c(t)$, variables, $v_i(t)$, and literals, $l_i(t)$.

Definition 5 (Dynamic Variable)**.** A dynamic variable is variable $x(t)$ such that,

$$x(t) \Leftrightarrow \begin{cases} x_0, & t < 0 \\ x, & t \geq 0 \end{cases}$$

where x_0 is an initial value of x.
For a system model $M(t)$, the propositional sentences describe *time-dependent* relations between variables. We express these relations by means of time delay $\delta \in \mathbb{R}^+$. We assume δ to be known and constant, i.e., not a function of time t. We also assume that each component has one characteristic propagation delay δ_c which coincides with the physical propagation time of that component. Let Δ be the set of all model time delays, $\Delta = \{\delta_1, \ldots, \delta_c, \ldots, \delta_{|C|}\}$.

Definition 6 (Dynamic FI Propositional Wff)**.** A dynamic FI propositional Wff is a formula over $l_1(t - \delta_1), \ldots, l_n(t-\delta_1), \ldots, l_1(t-\delta_c), \ldots, l_n(t-\delta_c), \ldots, l_1(t-\delta_{|C|}), \ldots, l_n(t-\delta_{|C|})$, and the standard Boolean connectives $\neg, \Leftrightarrow, \Rightarrow, \wedge, \vee$.

Note that time-dependent models are expressed only *relative* to t. To reason effectively with these models it is necessary that for some variables the initial condition at some point in time is known, i.e., observed. Furthermore we assume that variables keep their value after a delay unless specified otherwise. Thus, for example, if in a model a sentence consists of only one variable, $x(t-\delta)$, and it is the only sentence that involves x than x will be true indefinitely, starting at a time δ after t.

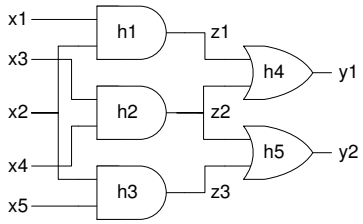


Figure 2: Polycell diagram.

We can express intervals by combining two delays, thus,

$$\begin{array}{l} \neg x_0 \\ y(t) \quad \Leftrightarrow \quad (x(t-\delta_1) \wedge \neg x(t-\delta_2)) \end{array}$$

defines y to be true for the interval between δ_1 and δ_2 after t and false otherwise. This is also useful for expressing the timing accuracy of observing y. State can be expressed by defining recurrent relations between a variable at time t and prior time $t - \delta$. For example,

$$\begin{array}{l} \neg v_0 \\ v(t) \quad \Leftrightarrow \quad \neg v(t - \frac{1}{2}) \end{array}$$

models a clock signal with period 1, starting at $true$ for $t = 0$.

Example Problems

We discuss two examples in which we use a time-dependent health to model propagation delays and, through recursion, state. We use these as leading examples throughout the rest of this paper. The first example is derived from the Polycell model (de Kleer & Williams 1987). This model consists of three multipliers and two adders. We use a Boolean model with three And gates and two Or gates as shown in Figure 2 and add dynamic behavior by introducing propagation delays. Only x and y variables are observable. The Polycell model is as follows,

$$\begin{array}{lcl} h_{a1}(t) & \Rightarrow & (z_1(t) \Leftrightarrow (x_1(t-\delta_{a1}) \wedge x_2(t-\delta_{a1}))) \\ h_{a2}(t) & \Rightarrow & (z_2(t) \Leftrightarrow (x_3(t-\delta_{a2}) \wedge x_4(t-\delta_{a2}))) \\ h_{a3}(t) & \Rightarrow & (z_3(t) \Leftrightarrow (x_2(t-\delta_{a3}) \wedge x_5(t-\delta_{a3}))) \\ h_{o1}(t) & \Rightarrow & (y_1(t) \Leftrightarrow (z_1(t-\delta_{o1}) \vee z_2(t-\delta_{o1}))) \\ h_{o2}(t) & \Rightarrow & (y_2(t) \Leftrightarrow (z_2(t-\delta_{o2}) \vee z_3(t-\delta_{o2}))) \end{array}$$

where $\delta_{a1}, \ldots, \delta_{o2}$ are real-valued propagation delays. We remark that the Polycell is *stateless* (combinatorial) as the circuit has no cycles.

The second example is a model of system *with* state: a Set/Reset (SR) latch as shown in Figure 3. When set, s is

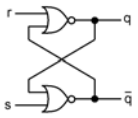


Figure 3: SR latch diagram.

$true$ the outputs q and $\overline{q}$ are latched $true$ and $false$ respectively, after a delay. For reset r, the outputs are opposite. This latching is implemented with recurrent relations that express state behavior for q and $\overline{q}$. The model is as follows.

$$\begin{aligned} h_1(t) &\Rightarrow (q(t) \Leftrightarrow \neg(r(t-\delta) \vee \overline{q}(t-\delta))) \\ h_2(t) &\Rightarrow (\overline{q}(t) \Leftrightarrow \neg(s(t-\delta) \vee q(t-\delta))) \end{aligned}$$

where δ is typically a very small propagation delay.

Discrete Numeric Expansion

A relatively straightforward method of converting a time-dependent problem is discrete numeric expansion of the time-dependent model. We use this method as reference for the more advanced factorization in the next section. First, the model with continuous time needs to be converted to one with discrete time. Thus, for all delays Δ we assume that there exists a sufficiently accurate integer representation $\Delta^{\mathbb{N}^+} = \left\{\delta_1^{\mathbb{N}^+}, \ldots, \delta_c^{\mathbb{N}^+}, \ldots, \delta_{|C|}^{\mathbb{N}^+}\right\} \subset \mathbb{N}^+$. We convert the continuous t to a discrete time k. Let M be defined as a dynamic Wff according to Definition 6. The numeric expansion method consists of the following consecutive steps (summarized in Algorithm 1).

1. Find the maximum discretization step, δ such that $\forall \delta_c^{\mathbb{N}^+} \in \Delta^{\mathbb{N}^+}, \frac{\delta_r^{\mathbb{N}^+}}{\delta} \in \mathbb{N}^+$. Thus, δ is the greatest common divisor of $\Delta^{\mathbb{N}^+}$.
2. Discretize by replacing t with $k = \frac{t}{\delta}$ and $\delta_c^{\mathbb{N}^+}$ with $\frac{\delta_c^{\mathbb{N}^+}}{\delta}$.

Next, we define a suitable discrete expansion range. We take the maximum propagation delay δ_{prop} which is the delay if the system itself would be considered as a component. Increasing this range would allow for diagnosis at more time steps. Note that diagnosis of recurrent components becomes more accurate with increased sequence of observations. Hence for models with recursion also the *quality* of diagnosis increases with this range. We continue the expansion as follows.

3. Find the maximum propagation delay δ_{prop}.
4. Expand the model by instantiating all components for $0 \leq k \leq \delta_{prop}$

For the Polycell example assume the following values: $\delta = \delta_{a1} = 1 = \frac{1}{2}\delta_{a2} = \frac{1}{3}\delta_{a3} = \delta_{o1} = \frac{1}{2}\delta_{o2}$. This leads to a maximum propagation delay of 5. By applying Algorithm 1

Algorithm 1 Discrete Numeric Expansion of a time-dependent model.

```
1: function ExpandModel(M, Δ)
       inputs: M, a model
               Δ, set containing all delays in M
       local variables: δ, discretization step
                        δ_prop, model propagation delay
                        k, discrete time
                        c, model component
2:     δ ← GreatestCommonDivisor(Δ)
3:     DiscretizeModel(M, δ)
4:     δ_prop ← GetMaxPropDelay(M)
5:     for all k ∈ [0, ..., δ_prop] do
6:         for all c ∈ M do
7:             Instantiate(c, k)
8:         end for
9:     end for
10: end function
```

we obtain.

$$\begin{aligned} \forall k &\in \{0, \ldots, 5\} \\ h_{a1}[k] &\Rightarrow (z_1[k] \Leftrightarrow (x_1[k-1] \wedge x_2[k-1])) \\ h_{a2}[k] &\Rightarrow (z_2[k] \Leftrightarrow (x_3[k-2] \wedge x_4[k-2])) \\ h_{a3}[k] &\Rightarrow (z_3[k] \Leftrightarrow (x_2[k-3] \wedge x_5[k-3])) \\ h_{o1}[k] &\Rightarrow (y_1[k] \Leftrightarrow (z_1[k-1] \vee z_2[k-1])) \\ h_{o2}[k] &\Rightarrow (y_2[k] \Leftrightarrow (z_2[k-2] \vee z_3[k-2])) \end{aligned}$$

Similar for the SR latch, if we assume $\delta = 1$, the maximum propagation delay becomes 2, and the expansion yields.

$$\begin{aligned} \forall k &\in \{0, 2\} \\ h_1[k] &\Rightarrow (q[k] \Leftrightarrow \neg(r[k-1] \vee \overline{q}[k-1])) \\ h_2[k] &\Rightarrow (\overline{q}[k] \Leftrightarrow \neg(s[k-1] \vee q[k-1])) \end{aligned}$$

The model is solvable by a classical algorithm.

Symbolic Factorization

We improve on the expansion by being selective, i.e., expanding only required variables, and doing it symbolically rather than numeric. This also means it is no longer required that Δ maps to $\mathbb{N}^+$. The goal is to factor all time-dependent variables out of the equations so that a traditional algorithm can solve the diagnosis problem without the need to reason in terms of time-dependent relations. To preserve the time-dependent constraints we symbolically expand the relations for variables in time, but only if the expanded relations are required to solve the remaining equations. Factoring out time dependencies provides a more efficient solution for the time-dependent diagnosis problem.

For example, consider two serial buffers with delay,

$$\begin{aligned} h_1(t) &\Rightarrow (o_1(t) \Leftrightarrow i(t-\delta)) \\ h_2(t) &\Rightarrow (o_2(t) \Leftrightarrow o_1(t-\delta)) \end{aligned}$$

We factor out all time-dependencies, by translating the first equation in time by δ,

$$h_1(t-\delta) \Rightarrow (o1(t-\delta) \Leftrightarrow i(t-\delta-\delta))$$

as $o_1(t-\delta)$ is used in the second equation. Next, we replace all time-dependent variables with symbols according to,

$$\begin{aligned} h_1 &\Rightarrow (o_1 \Leftrightarrow i_\delta) \\ h_2 &\Rightarrow (o_2 \Leftrightarrow o_{1\delta}) \\ h_{1\delta} &\Rightarrow (o_{1\delta} \Leftrightarrow i_{\delta\delta}) \end{aligned}$$

$$\begin{aligned} h_1 &\Leftrightarrow h_1(t) \\ o_1 &\Leftrightarrow o_1(t) \\ i_\delta &\Leftrightarrow i(t-\delta) \\ h_2 &\Leftrightarrow h_2(t) \\ o_2 &\Leftrightarrow o_2(t) \\ o_{1\delta} &\Leftrightarrow o_1(t-\delta) \\ h_{1\delta} &\Leftrightarrow h_1(t-\delta) \\ o_{1\delta} &\Leftrightarrow o_1(t-\delta) \\ i_{\delta\delta} &\Leftrightarrow i(t-2\delta) \end{aligned}$$

which effectively factorizes the model in a static part, defining system structure and behavior (first three equations), and a dynamic part (remaining equations) which merely links the new symbols to the observables (i,o) or health variables (h). Thus, symbolic factorization takes the following consecutive steps.

1. Find all dynamic variables $v(t)$ that are shared between components and have a delay.
2. Instantiate the component that determines $v(t)$ for all delays it is used with by the other components.
3. Repeat step 2 until an input with maximum propagation delay δ_{prop} is used. This is similar to the range size with the numeric expansion method and can also be increased for diagnosis of more time steps or, in case of recursion, better quality of diagnosis.
4. Replace all variables with static ones.

The above is summarized in Algorithm 2. Compared to the *diagnosis* algorithms that are subsequently applied, the computational complexity of symbolic factorization is negligible.

Applying Algorithm 2 to the Polycell model leads to following results. Step 1 through 3 yield the following results.

$$\begin{aligned} h_{a1}(t) &\Rightarrow (z_1(t) \Leftrightarrow (x_1(t-\delta_{a1}) \wedge x_2(t-\delta_{a1}))) \\ h_{a2}(t) &\Rightarrow (z_2(t) \Leftrightarrow (x_3(t-\delta_{a2}) \wedge x_4(t-\delta_{a2}))) \\ h_{a3}(t) &\Rightarrow (z_3(t) \Leftrightarrow (x_2(t-\delta_{a3}) \wedge x_5(t-\delta_{a3}))) \\ h_{o1}(t) &\Rightarrow (y_1(t) \Leftrightarrow (z_1(t-\delta_{o1}) \vee z_2(t-\delta_{o1}))) \\ h_{o2}(t) &\Rightarrow (y_2(t) \Leftrightarrow (z_2(t-\delta_{o2}) \vee z_3(t-\delta_{o2}))) \end{aligned}$$

$$\begin{aligned} h_{a1}(t-\delta_{o1}) &\Rightarrow (z_1(t-\delta_{o1}) \Leftrightarrow (x_1(t-\delta_{a1}-\delta_{o1}) \wedge \\ &\quad x_2(t-\delta_{a1}-\delta_{o1}))) \\ h_{a2}(t-\delta_{o1}) &\Rightarrow (z_2(t-\delta_{o1}) \Leftrightarrow (x_3(t-\delta_{a2}-\delta_{o1}) \wedge \\ &\quad x_4(t-\delta_{a2}-\delta_{o1}))) \\ h_{a2}(t-\delta_{o2}) &\Rightarrow (z_2(t-\delta_{o2}) \Leftrightarrow (x_3(t-\delta_{a2}-\delta_{o2}) \wedge \\ &\quad x_4(t-\delta_{a2}-\delta_{o2}))) \\ h_{a3}(t-\delta_{o2}) &\Rightarrow (z_3(t-\delta_{o2}) \Leftrightarrow (x_2(t-\delta_{a3}-\delta_{o2}) \wedge \\ &\quad x_5(t-\delta_{a3}-\delta_{o2}))) \end{aligned}$$

Algorithm 2 Symbolic factorization of a time-dependent model.

```
1: function FACTORIZEMODEL(M)
       inputs: M, a model
       local variables: V, shared variables
                        t, continuous time
                        v(t), time-dependent variable
                        δ, delay
                        v_δ, static variable
                        δ_prop, model propagation delay
                        δ_c, component delay
2:     V ← SHAREDVARIABLES(M)
3:     δ_prop ← GETMAXPROPDELAY(M)
4:     repeat
5:         for all v(t) ∈ V do
6:             for all v(t − δ) ∈ M do
7:                 INSTANTIATE(c, v(t − δ))
8:                 δ_c,max ←MAX(δ_c,max, δ_c)
9:             end for
10:        end for
11:    until δ_c,max = δ_prop
12:    for all v(t − δ) ∈ M do
13:        REPLACE(v(t − δ), v_δ)
14:    end for
15: end function
```

In Step 4 we factor out all time dependencies, together these form the dynamic model.

$$\begin{aligned} h_{a1} &\Rightarrow (z_1 \Leftrightarrow (x_{1\delta_{a1}} \wedge x_{2\delta_{a1}})) \\ h_{a2} &\Rightarrow (z_2 \Leftrightarrow (x_{3\delta_{a2}} \wedge x_{4\delta_{a2}})) \\ h_{a3} &\Rightarrow (z_3 \Leftrightarrow (x_{2\delta_{a3}} \wedge x_{5\delta_{a3}})) \\ h_{o1} &\Rightarrow (y_1 \Leftrightarrow (z_{1\delta_{o1}} \vee z_{2\delta_{o1}})) \\ h_{o2} &\Rightarrow (y_2 \Leftrightarrow (z_{2\delta_{o2}} \vee z_{3\delta_{o2}})) \\ h_{a1\delta_{o1}} &\Rightarrow (z_{1\delta_{o1}} \Leftrightarrow (x_{1\delta_{a1}\delta_{o1}} \wedge \\ &\quad x_{2\delta_{a1}\delta_{o1}})) \\ h_{a2\delta_{o1}} &\Rightarrow (z_{2\delta_{o1}} \Leftrightarrow (x_{3\delta_{a2}\delta_{o1}} \wedge \\ &\quad x_{4\delta_{a2}\delta_{o1}})) \\ h_{a2\delta_{o2}} &\Rightarrow (z_{2\delta_{o2}} \Leftrightarrow (x_{3\delta_{a2}\delta_{o2}} \wedge \\ &\quad x_{4\delta_{a2}\delta_{o2}})) \\ h_{a3\delta_{o2}} &\Rightarrow (z_{3\delta_{o2}} \Leftrightarrow (x_{2\delta_{a3}\delta_{o2}} \wedge \\ &\quad x_{5\delta_{a3}\delta_{o2}})) \end{aligned}$$

Again, the second part links the new symbols to the original time-dependent variables,

$$\begin{aligned} z_1 &\Leftrightarrow z_1(t) \\ h_{a1} &\Leftrightarrow h_{a1}(t) \\ x_{1\delta_{a1}} &\Leftrightarrow x_1(t-\delta_{a1}) \\ x_{2\delta_{a1}} &\Leftrightarrow x_2(t-\delta_{a1}) \\ &\vdots \end{aligned}$$

$$
\begin{aligned}
h_{o2} &\Leftrightarrow h_{o2}(t) \\
y_2 &\Leftrightarrow y_2(t) \\
h_{a2\delta_{o2}} &\Leftrightarrow h_{a2}(t-\delta_{o2}) \\
x_{3\delta_{a2}\delta_{o2}} &\Leftrightarrow x_3(t-\delta_{a2}-\delta_{o2}) \\
x_{4\delta_{a2}\delta_{o2}} &\Leftrightarrow x_4(t-\delta_{a2}-\delta_{o2}) \\
h_{a2\delta_{o2}} &\Leftrightarrow h_{a2}(t-\delta_{o2}) \\
h_{a3\delta_{o2}} &\Leftrightarrow h_{a3}(t-\delta_{o2}) \\
x_{2\delta_{a3}\delta_{o2}} &\Leftrightarrow x_2(t-\delta_{a3}-\delta_{o2}) \\
x_{5\delta_{a3}\delta_{o2}} &\Leftrightarrow x_5(t-\delta_{a3}-\delta_{o2}) \\
h_{a3\delta_{o2}} &\Leftrightarrow h_{a3}(t-\delta_{o2})
\end{aligned}
$$

For the SR latch the results are as follows.

$$
\begin{aligned}
h_1 &\Rightarrow (q \Leftrightarrow \neg(r_\delta \vee \overline{q}_\delta)) \\
h_2 &\Rightarrow (\overline{q} \Leftrightarrow \neg(s_\delta \vee q_\delta)) \\
h_{1\delta} &\Rightarrow (q_\delta \Leftrightarrow \neg(r_{\delta\delta} \vee \overline{q}_{\delta\delta})) \\
h_{2\delta} &\Rightarrow (\overline{q}_\delta \Leftrightarrow \neg(s_{\delta\delta} \vee q_{\delta\delta}))
\end{aligned}
$$

$$
\begin{aligned}
h_1 &\Leftrightarrow h_1(t) \\
q &\Leftrightarrow q(t) \\
r_\delta &\Leftrightarrow r(t-\delta) \\
\overline{q}_\delta &\Leftrightarrow \overline{q}(t-\delta) \\
&\vdots \\
h_{2\delta} &\Leftrightarrow h_2(t-\delta) \\
\overline{q}_\delta &\Leftrightarrow \overline{q}(t-\delta) \\
s_{\delta\delta} &\Leftrightarrow s(t-2\delta) \\
q_{\delta\delta} &\Leftrightarrow q(t-2\delta)
\end{aligned}
$$

In each case, the first part of the model is solved with a standard diagnosis algorithm while the second part has a trivial solution.

Implementation

We have validated our approach using LYDIA (Pietersma, Feldman, & van Gemund 2006) as MBD implementation. LYDIA (Language for sYstems DIAgnosis) is a language aimed at model-based reasoning. Currently a number of diagnosis algorithms have been implemented in an accompanying tool set[1]. The results in this paper were obtained with a diagnosis engine that uses Conflict-directed A* (Williams & Ragno 2004). We have also created prototype implementations of the expansion and factorization algorithms (Algorithms 1 and 2). Both operate on an intermediate representation of the LYDIA model and create an expanded and factorized LYDIA model, respectively. Table 1 lists the LYDIA subset as used in this paper. Propagation delays are implemented with a VHDL-like "after" language construct.

Listing 1 shows the LYDIA model for the Polycell example. The AndGate and OrGate component types both have a propagation delay. The AndGate instances a1, a2, a3 have delays d_a1, d_a2, d_a3 and the OrGate instances o1, o2 have delays d_o1, d_o2. The result of the expansion algorithm is shown in Listing 2. The result of the symbolic factorization is shown in Listing 3.

[1] Available at http://fdir.org/lydia/.

LYDIA syntax	description
system	component
bool	Boolean variable
real	real variable
not, **=**, **=>**, **and**, **or**, **nor**	$\neg, \Leftrightarrow, \Rightarrow, \wedge, \vee, \downarrow$
[...]	array index
forall	$\forall$
after	time delay

Table 1: LYDIA syntax

Listing 1: Dynamic Polycell model.

```
system AndGate (
  bool h, x1, x2, y, real d )
{
  h => ( y = (x1 and x2) after d );
}

system OrGate (
...
system Polycell (
  bool h_a1, h_a2, h_a3, h_o1, h_o2,
  x1, x2, x3, x4, x5, y1, y2,
  real d_a1, d_a2, d_a3, d_o1, d_o2 )
{
  bool z1, z2, z3;
  system AndGate a1, a2, a3;
  system OrGate o1, o2;

  a1 ( x1, x2, z1, d_a1 );
  a2 ( x3, x4, z2, d_a2 );
  a3 ( x2, x5, z3, d_a3 );
  o1 ( z1, z2, y1, d_o1 );
  o2 ( z2, z3, y2, d_o2 );
}
```

Listing 2: Expanded Polycell model.

```
system AndGate_ (
  bool h, x1, x2, y)
{
  h => ( y = x1 and x2);
}

...
  system AndGate_ a1, a2, a3;
  system OrGate_ o1, o2;

  forall (k in 0..5) {
    a1[k] ( h_a1[k], x1[k], x2[k], z1[k] );
    a2[k] ( h_a2[k], x3[k], x4[k], z2[k] );
    a3[k] ( h_a3[k], x2[k], x5[k], z3[k] );
    o1[k] ( h_o1[k], z1[k], z2[k], y1[k] );
    o2[k] ( h_o2[k], z2[k], z3[k], y2[k] );
  }
}
```

Listing 3: Factorized Polycell model.

```
...
  system AndGate_ a1, a2, a3, a1_do1, a2_do1,
  a2_do2, a3_do2;
  system OrGate_ o1, o2;

  x1_da1 = x1 after da1;
...
  h_a1_do1 = h_a1 after do1;
  x1_da1_do1 = x1 after (da1 + do1);
  x2_da1_do1 = x2 after (da1 + do1);
...
  a1 ( h_a1, x1_da1, x2_da1, z1 );
  a2 ( h_a2, x3_da2, x4_da2, z2 );
  a3 ( h_a3, x2_da3, x5_da3, z3 );
  a1_do1 ( h_a1_do1, x1_da1_do1,
    x2_da1_do1, z1_do1 );
  a2_do1 ( h_a2_do1, x3_da2_do1,
    x4_da2_do1, z2_do1 );
  a2_do2 ( h_a2_do2, x3_da2_do2,
    x4_da2_do2, z2_do2 );
  a3_do2 ( h_a3_do2, x2_da3_do2,
    x5_da3_do2, z2_do2 );
  o1 ( h_o1, z1_do1, z2_do1, y1 );
  o2 ( h_o2, z2_do2, z3_do2, y2 );
...
```

Listing 4 and Listing 5 respectively show the expanded and factorized LYDIA model of the SR latch.

Listing 4: Expanded SR latch model.

```
...
  forall (k in 0..2) {
    h1[k] => ( q[k] = r[k-1] nor _q[k-1] );
    h2[k] => ( _q[k] = s[k-1] nor q[k-1] );
  }
...
```

Listing 5: Factorized SR latch model.

```
...
  _q_d = _q after d;
  q_d = q after d;
  h1 => (q = r_d nor _q_d);
  h2 => (_q = s_d nor q_d);
...
```

Performance Evaluation

We have evaluated the performance of both methods with models based on circuits in the ISCAS-85 benchmark. For this purpose we have modified the available LYDIA ISCAS-85 models by adding propagation delays to components at the gate level. We converted these models using both Algorithms 1 and 2, and diagnosed them for single fault scenarios using an existing Conflict Directed A* LYDIA engine on a 3 GHz Pentium IV CPU.

Table 2 presents the results. The first column states the circuit name. The number of components in the original (static) model is indicated with $|C|$. The maximum propagation delay is indicated with δ_{prop}. For the expanded (exp.) and factorized (fac.) models the number of health variables is denoted by $|C|_{exp.}$ and $|C|_{fac.}$ respectively. The reduction (red.) in the number of health variables that we obtain with symbolic factorization compared to numeric expansion is given in percentages in the sixth column. The diagnosis time is denoted by T and given for both methods in ms. The last column gives the speedup in diagnosis time for the factorization method.

As expected, the symbolic factorization yields a significant reduction in both the number of health variables (33-78%) and, more importantly, the speed-up in diagnosis time (2.32 to 175.76). Due to the inefficiency of the numeric expansion and resulting memory exhaustion, the diagnosis engine could not cope with the last three models listed. A comparison of diagnosis times for these models could therefore not be made.

Discussion

As is visible from Algorithm 1, the listed conversion results, and the experiments, the expansion method is not very efficient. The number of health variables in the resulting static model grows linearly with the number of expansion time steps, δ_{prop}. As the diagnosis time grows exponentially with the number of health variables, this makes the expansion method inefficient for systems with large variability in delay lengths. Consider for example a system with 10 sequential components of which 9 have a delay of 10 time steps and one with a delay of 1 time step. This particular system is then expanded for $(\delta_{prop} - \max(\Delta))/\delta = (91 - 10)/1 = 81$ time steps leading to an equal amount of health variables. Systems that integrate behavior across different physical domains, e.g., electro-mechanical, tend to have this variability in delays. Therefore, the expansion method is not suited for these types of systems.

The factorization method is much more efficient than the expansion method since the number of static health variables is merely dependent on the number of shared variables and the number of components with different delays that use it. Since most engineering systems display some form of hierarchy the number of interconnections is relatively limited and, accordingly, the number of static health variables and diagnosis time. In contrast to the numeric expansion, the complexity of the factorization algorithm is not affected by the actual values of the delays. The factorization approach works for systems with constant propagation delays. However, for system with recurrent relations, i.e., feedback, the quality of diagnosis is critically dependent on the duration of the observation window and, as always, the number of variables observed.

Compared to Livingstone, we obtain similar results, though with different notation, under the assumption of synchrony. Assumption of synchrony in both our approaches means only *one* delay value for the whole model. In Livingstone state is implemented with the next operator $\bigcirc$. For example, $x => \bigcirc(y)$ is equivalent with $x(t - \delta) \Rightarrow y(t)$. Again as mentioned above, we believe non-synchrony to be an important requirement for modeling of multi-disciplinary systems. The synchronous state transitions may lead to a state explosion when modeling systems with different time

model	$\|C\|$	δ_{prop}	$\|C\|_{exp.}$	$\|C\|_{fac.}$	red. [%]	$T_{exp.}$ [ms]	$T_{fac.}$ [ms]	speedup
74181	62	2	124	83	33	738.2	4.2	175.76
74182	19	3	57	34	40	557.7	11.3	49.35
74283	40	3	120	60	50	8.9	2.3	3.87
74L85	41	2	82	53	35	4.4	1.9	2.32
c432	146	3	438	178	59	104.5	14.9	7.01
c499	202	2	404	210	48	87.6	22.6	3.88
c1355	513	2	1026	522	49	-	114.8	-
c1908	251	8	2008	451	78	-	99.0	-
c2670	982	2	1964	1037	47	-	579.1	-

Table 2: Experimental results for ISCAS-85 models.

resolutions since the largest delays need to be expressed in terms of the smallest. This is similar to the numeric expansion method. The factorization approach allows for a more natural way of modeling dynamic systems with arbitrary delays and does not suffer from this problem as all delays are factored out. Moreover in our approach the existing algorithm is oblivious to state and deals with the converted dynamic model just like any other.

As a final remark, note that the speedups as presented in Table 2 constitute a lower bound as for all the gates in the models the *same* propagation delay has been used. In reality, therefore, the complexity reduction is even greater.

Conclusions

We have shown that models with propagation delays can be effectively used for the diagnosis of dynamic system, including systems with state. Factoring out propagation delays and solving the resulting models with existing algorithms has many advantages. The user can define arbitrary real-valued delays which allows him to realistically model systems from different engineering domains without having to resort to temporal abstractions. Compared to numeric expansion, factorization is computationally much more efficient as the model is only expanded where necessary, i.e., where additional time-dependent relations are required for consistent diagnosis. This is illustrated with significant speedup in diagnosis time. Furthermore, the efficiency of factorization does not suffer from a variability in propagation delay, in contrast to the expansion method. Consequently, the speedups involved will significantly increase when propagation delays differentiate per component type

References

Abdelwahed, S.; Karsai, G.; and Biswas, G. 2005. A consistency-based robust diagnosis approach for temporal causal systems. In Dearden, R., and Narasimhan, S., eds., *Proceedings of the Sixteenth International Workshop on Principles of Diagnosis (DX-05), Monterey California, USA*, 73–79.

Armengol, J.; Vehi, J.; Trave-Massuyes, L.; and Sainz, M. 2001. Application of multiple sliding time windows to fault detection based on interval models.

Dague, P.; Jehl, O.; and Taillibert, P. 1990. An interval propagation and conflict recognition engine for diagnosing continuous dynamic systems. In *Proceedings of the international workshop on Expert systems in engineering : Principles and applications*, 16–31. New York, NY, USA: Springer-Verlag New York, Inc.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *JAI* 32(1):97–130.

de Kleer, J., and Williams, B. C. 1992. Diagnosis with behavioral modes. 124–130.

Dvorak, D., and Kuipers, B. 1989. Model-based monitoring of dynamic systems. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, 1238–1243. San Mateo, CA: Morgan Kaufmann.

Glass, B. J.; Erickson, W. K.; and Swanson, K. J. 1991. Texsys: A large scale demonstration of model-based real-time control of a space station subsystem. In *Proc. of the Seventh Conference on Artificial Intelligence Applications CAIA-92 (Volume I: Technical Papers)*, 378–384.

Grastien, A.; Cordier, M.-O.; and Largout, C. 2005. Incremental diagnosis of discrete-event systems. In Dearden, R., and Narasimhan, S., eds., *Proceedings of the Sixteenth International Workshop on Principles of Diagnosis (DX-05), Monterey California, USA*, 119–124.

Guckenbiehl, T., and Schafer-Richter, G. 1990. Sidia: extending prediction based diagnosis to dynamic models. In *Proceedings of the international workshop on Expert systems in engineering : Principles and applications*, 53–68. New York, NY, USA: Springer-Verlag New York, Inc.

Hamscher, W. C. 1991. Modeling digital circuits for troubleshooting. *Artif. Intell.* 51(1-3):223–271.

Kurien, J., and Nayak, P. P. 2000. Back to the future for consistency-based trajectory tracking. In *AAAI/IAAI*, 370–377.

Pietersma, J.; Feldman, A.; and van Gemund, A. J. 2006. Modeling and compilation aspects of fault diagnosis complexity. In of Directors, A. B., ed., *2006 IEEE AUTOTESTCON IEEE Systems Readiness Technology Conference Proceedings*, 502–508. IEEE.

Reiter, R. 1987. A theory of diagnosis from first principles. In Ginsberg, M. L., ed., *Readings in Nonmonotonic Reasoning*. Los Altos, California: Kaufmann. 352–371.

Williams, B. C., and Nayak, P. P. 1999. A model-based approach to reactive self-configuring systems. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Williams, B., and Ragno, R. 2004. Conflict-directed A* and its role in model-based embedded systems. *JDAM*.

Williams, B. C. 1990. Doing time: putting qualitative reasoning of firmer ground. In *Readings in qualitative reasoning about physical systems*, 353–360. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Advanced Diagnostics and Prognostics Testbed

Scott Poll, Ann Patterson-Hine, Joe Camisa, David Garcia[1], David Hall[1], Charles Lee[2], Ole J. Mengshoel[3], Christian Neukom[1], David Nishikawa, John Ossenfort[2], Adam Sweet[1], Serge Yentus[1]
[1]QSS Group, Inc., a subsidiary of Perot Systems Government Services; [2]SAIC; [3]RIACS
NASA Ames Research Center
M/S 269-1, Moffett Field, CA 94035
{Scott.Poll, Ann.Patterson-Hine}@nasa.gov

Indranil Roychoudhury, Matthew Daigle, Gautam Biswas, Xenofon Koutsoukos
Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science,
Vanderbilt University, Nashville, TN 37235
gautam.biswas@vanderbilt.edu

Abstract

Researchers in the diagnosis community have developed a number of promising techniques for system health management. However, realistic empirical evaluation and comparison of these approaches is often hampered by a lack of standard data sets and suitable testbeds. In this paper we describe the Advanced Diagnostics and Prognostics Testbed (ADAPT) at NASA Ames Research Center. The purpose of the testbed is to measure, evaluate, and mature diagnostic and prognostic health management technologies. This paper describes the testbed's hardware, software architecture, and concept of operations. A simulation testbed that accompanies ADAPT, and some of the diagnostic and decision support approaches being investigated are also discussed.

Introduction

Automated methods for diagnosing problems with system behavior are commonplace in automobiles, copiers, and many other consumer products. Applying advanced diagnostic techniques to aerospace systems, especially aerospace vehicles with human crews, is much more challenging. The low probability of component and subsystem failure, the cost of verification and validation, the difficulty of selecting the most appropriate diagnostic technology for a given problem, and the lack of large-scale diagnostic technology demonstrations increase the complexity of these applications. To meet these challenges, NASA Ames Research Center has developed the Advanced Diagnostic and Prognostic Testbed with the following goals in mind:

(i) Provide a technology-neutral basis for testing and evaluating diagnostic systems, both software and hardware,

(ii) Provide the capability to perform accelerated testing of diagnostic algorithms by manually or algorithmically inserting faults,

(iii) Provide a real-world physical system such that issues that might be disregarded in smaller-scale experiments and simulations are exposed – "the devil is in the details,"

(iv) Provide a stepping stone between pure research and deployment in aerospace systems, thus create a concrete path to maturing diagnostic technologies, and

(v) Develop analytical methods and software architectures in support of the above goals.

NASA missions have always provided challenging problem domains for model-based diagnosis researchers. Early demonstrations (Williams *et al.*, 1998) were typically limited to particular subsystems or components, and were run in "shadow-mode," in parallel with the conventional control and fault protection software. However, the increasing complexity of aerospace systems has made advanced diagnostic capabilities a necessity to ensure the reliability and safety of missions. Although they are well-suited to address the needs of aerospace missions, many of the techniques that fall under the "model-based diagnosis" umbrella face opposition from project managers, who may be reluctant to consider techniques with limited flight experience.

We have developed the Advanced Diagnostics and Prognostics Testbed at NASA Ames Research Center to enable maturation of advanced fault management techniques. ADAPT provides a representative physical domain that serves as a benchmark for performance assessment and comparison of algorithms and concepts that enable automated diagnosis of complex systems exhibiting hybrid, i.e., both continuous and discrete, behavior. The testbed documentation includes engineering schematics, component descriptions, fault descriptions, and system engineering analyses, such as functional models, Failure Modes and Effects Analysis, and Testability Analysis. Additional information includes a simulation and an Application Programming Interface (API) to facilitate integrating diagnostic applications with the testbed. We will provide the documentation, simulation model, API, as well as nominal and faulty testbed data to researchers in the diagnosis community.

The process of developing advanced diagnostic applications is just as important to the verification and validation of these systems as the specific algorithms that are applied. ADAPT establishes a problem domain with known fault signatures and the capability to inject failures during operation of the testbed. By implementing and testing different

diagnostic approaches on ADAPT, we aim to improve performance assessment methods and the comparison of diverse health management strategies.

This paper presents a description of ADAPT that includes its hardware and software architectures, the concept of operations, and a simulation testbed that emulates the real system. We also discuss some of the diagnostic approaches that are currently being developed on the testbed.

Testbed Description

The ADAPT lab is shown in Figure 1. The equipment racks in the background can generate, store, distribute, and monitor electrical power. The initial testbed configuration functionally represents an exploration vehicle's Electrical Power System (EPS). The EPS can deliver AC (Alternating Current) and DC (Direct Current) power to loads, which in an aerospace vehicle would include subsystems such as the avionics, propulsion, life support, and thermal management systems. A data acquisition and control system sends commands to and receives data from the EPS. The testbed operator stations are integrated into a software architecture that allows for nominal and faulty operations of the EPS, and includes a system for logging all relevant data to assess the performance of the health management applications. The following sections describe the testbed hardware, diagnostic challenges, concept of operations, and software.

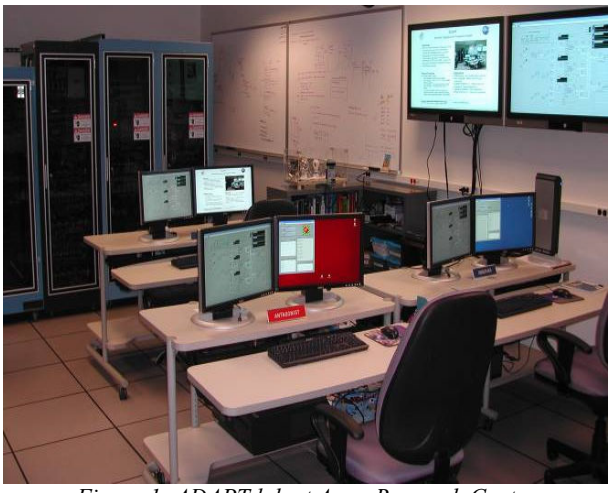


Figure 1: ADAPT lab at Ames Research Center.

Hardware Subsystems

Figure 2 depicts ADAPT's major system components and their interconnections. Three power generation sources are connected to three sets of batteries, which in turn supply two load banks. Each load bank has provisions for 6 AC loads and 2 DC loads.

Power Generation. The three sources of power generation include two battery chargers and a photovoltaic module. The battery chargers are connected to appropriate wall outlets through relays. Two metal halide lamps supply the light energy for the photovoltaic module. The three power generation sources can be interchangeably connected to the three batteries. Hardware relay logic prevents connecting one charge source to more than one battery at the same time, and from connecting one charging circuit to another charging circuit.

Power Storage. Three sets of batteries are used to store energy for operation of the loads. Each "battery" consists of two 12-volt sealed lead acid batteries connected in series to produce a 24-volt output. Two battery sets are rated at 100 amp-hrs and the third set is rated at 50 amp-hrs. The batteries and the main circuit breakers are placed in a ventilated cabinet that is physically separated from the equipment racks; however, the switches for connecting the batteries to the upstream chargers or downstream loads are located in the equipment racks.

Power Distribution. Electromechanical relays are used to route the power from the sources to the batteries and from the batteries to the AC and DC loads. All relays are the normally-open type. An inverter converts the 24-volt DC

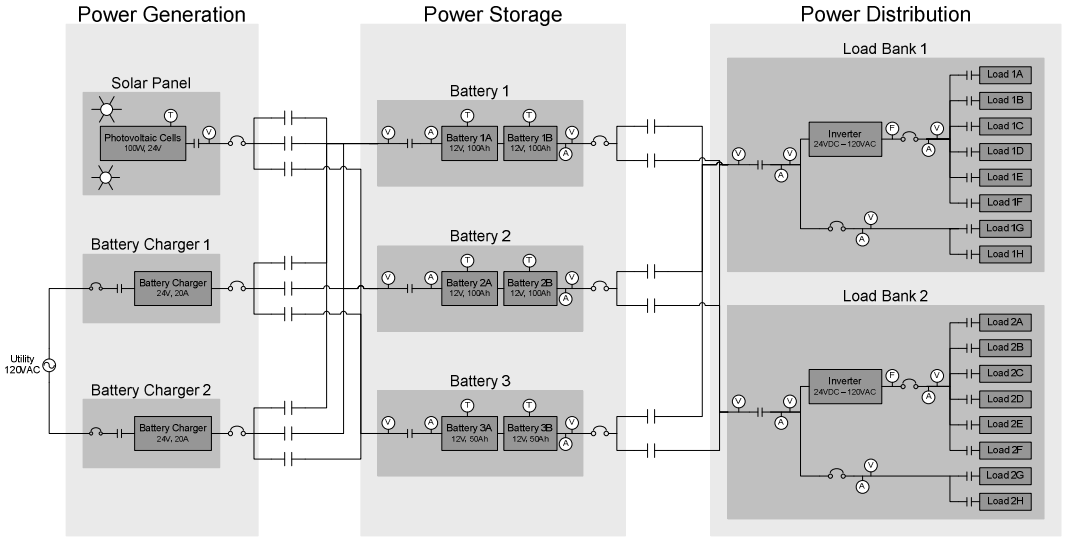


Figure 2: Testbed components and interconnections.

battery input to a 120-volt rms AC output. Circuit breakers are located at various points in the distribution network to prevent overcurrents from causing unintended damage to the system components.

Control and Monitoring. Testbed data acquisition and control use National Instrument's LabVIEW software and CompactFieldPoint (cFP) hardware. Table 1 lists the modules that are inserted into the two identical backplanes. The instrumentation allows for monitoring of voltages, currents, temperatures, switch positions, light intensities, and AC frequencies, as listed in Table 2.

Table 1: Testbed backplane modules.

Module	Description	Channels
cFP-2000	Real-time Ethernet Module	NA
cFP-DI-301	Digital Input Module	16 (x2)
cFP-DO-401	Digital Output Module	16 (x2)
cFP-AI-100	Analog Input Module	8
cFP-AI-102	Analog Input Module	8 (x2)
cFP-RTD-122	RTD Input Module	8

Table 2: Testbed instrumentation.

Sensed Variable	Number of Sensors
Voltage	22
Current	12
Temperature	15
Relay Position	41
Circuit Breaker Position	17
Light Intensity	3
AC Frequency	2

Diagnosis Challenges

The ADAPT testbed offers a number of challenges to health management applications. The electrical power system shown in Figure 2 is a hybrid system with multiple system configurations made possible by switching among the generation, storage, and distribution units. Timing considerations and transient behavior must be taken into account when designing diagnosis algorithms. When power is input to the inverter there is a delay of a few seconds before power is available at the output. For some loads, there is a large current transient when the device is turned on. As shown in Figure 3, system voltages and currents depend on the loads attached, and noise in the sensor data becomes more pronounced as more loads are added. Due to the low probabilities of failure, seeding/inserting faults is needed. Through an antagonist function described in the next section, it is possible to inject multiple faults into the testbed.

Concept of Operations

Unlike many other testbeds, the primary articles under test in ADAPT are the health management systems, not the physical devices of the testbed. To operate the testbed in a way that facilitates the study of health management technologies, the following operator roles are defined:

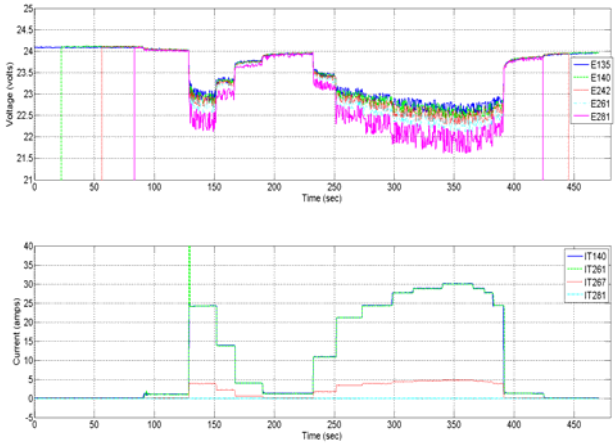


Figure 3: Sample testbed voltages (top) and currents (bottom) for a battery discharging to loads.

- *User* – who simulates the role of a crew member or pilot operating and maintaining the testbed subsystems.
- *Antagonist* – who injects faults into the subsystem either manually, remotely through the *Antagonist* console, or automatically through software scripts.
- *Observer* – who records the experimental data and notes how the *User* responds to the faults injected by the *Antagonist*. The *Observer* also serves as the safety officer during all tests and can initiate an emergency stop.

During an experiment, the *User* is responsible for controlling and monitoring the EPS and any attached loads that are required to accomplish a mission. The *Antagonist* disrupts system operations by injecting one or more faults unbeknownst to the *User*. The *User* may use output from a health management application (test article) to determine the state of the system and choose an appropriate recovery action. The *Observer* records the interactions and measures the effectiveness of the test article.

The testbed has two primary goals: (i) performance analysis of, and comparisons among, different test articles, and (ii) running of system studies. With the hardware and the supporting software infrastructure described in a subsequent section, experiments may be conducted using a variety of test articles to evaluate different health management technologies. The test articles may be connected to different interface concepts for presenting health management information to the human *User*, to study how the person performs in managing system operations.

Software

The testbed software model supports the concept of operations that includes the previously-mentioned operational roles of the *User* (USR), *Antagonist* (ANT), *Observer* (OBS), and *Test Article* (TA), along with the *Logger* (LOG) and the *Data Acquisition* (DAQ) roles. The *Logger* collects and saves all communication between the various components. The DAQ computer interfaces with the National Instruments data acquisition and control system. It sends command data to the testbed via the appropriate

backplane modules and receives testbed sensor data from other modules.

The underlying data communication is implemented using a publish/subscribe model in which data from publishers are routed to all subscribers registering an interest in a data topic. To enforce testing protocols and ensure the integrity of the data, filters based on role and location limit the topics for which data can be produced or consumed. Table 3 lists the message topics and the topic publishers and subscribers.

Table 3: Publish/subscribe topics.

Topic	Publisher	Subscriber
Sensor Data	DAQ	ANT,OBS,LOG
Antagonist Data	ANT	TA,USR,LOG
User Command	USR,TA	TA,ANT,OBS,LOG
Antagonist Command	ANT	DAQ,OBS,LOG
User Message	USR	OBS,LOG,ANT
Note	OBS	LOG,ANT
Fault Data	TA	USR,OBS,LOG
Diagnostics	TA	USR,OBS,LOG
Experiment Control	OBS	LOG

The following constraints are enforced on the various system components when they are operating on the ADAPT computer network:

- The *DAQ* can only read command data sent by the Antagonist and only sends sensor data which it gets directly from the instrumentation I/O subsystem. When no faults are injected, the *Antagonist* commands are the same as the *User* commands. The *DAQ* software can run only on the *DAQ* computer. It is the only software that connects directly to the instrumentation I/O subsystem.
- The *Antagonist* can only read sensor data sent by the *DAQ* and command data sent by *Test Articles* and *Users*. It forwards sensor data, which it may have modified by fault injection. It also sends command data, which are read by the *DAQ* and the *Logger*.
- The *Test Article* and the *User* cannot see *DAQ* sensor data. They have access only to *Antagonist*-generated sensor data, which are identical to *DAQ* sensor data when no faults are injected. The *Test Article* and *User* cannot read text data (a *Note*) sent by the *Observer*. The *Test Article* can read *User* commands and *Antagonist* data. It can send diagnostics data and *User* commands.
- The *Observer* sends an experiment control record to initiate a testbed experiment. It also sends text data to describe observations of the ongoing experiment. The *Observer* cannot send any data other than control and text data.
- The *Logger* reads all data sent over the ADAPT network. The *Logger* assigns a unique experiment ID for each new experiment.

A test article may be integrated with the testbed by installing the application on one of the ADAPT computers or by connecting a computer with the test article application to a preconfigured auxiliary system that acts as a gateway to the ADAPT network. A test article uses an API to subscribe to commands and data, and publish diagnosis results to the ADAPT message server. Java and C++ interfaces are supported.

VIRTUAL ADAPT: Simulation Testbed

We are also developing a high-fidelity simulation testbed that emulates the ADAPT hardware for running offline health management experiments. This environment, called VIRTUAL ADAPT, provides identical interfaces to the application system modules through wrappers to the ADAPT network. The physical components of the testbed, i.e., the chargers, the batteries, relays, and the loads, are replaced by simulation modules that generate the same dynamic behaviors as the hardware test bed. Also, like the actual hardware, VIRTUAL ADAPT subscribes to antagonist commands and publishes corresponding sensor data. As a result, application systems developed on VIRTUAL ADAPT can be run directly on ADAPT, and vice versa. Therefore, applications can be developed and tested using VIRTUAL ADAPT, and then run as test articles on the actual system. The simulation environment also provides for precise repetition of different operational scenarios, and this allows for more rigorous testing and evaluation of different diagnostic and prognostic algorithms.

In order to mirror the real testbed, we have addressed several issues that include (i) one-to-one component modeling, (ii) replicating the dynamic behavior of the hardware components, (iii) matching the possible configurations, and (iv) facilitating the running of diagnosis and prognosis experiments. The following sections provide a more detailed description of our approach to addressing these issues.

Component-Oriented Modeling

Our approach to component-oriented compositional modeling is based on a top-down process, where we first capture the structural description of the system in terms of the components and their connectivity. Component models replicate the component's dynamic behaviors for different configurations. The connectivity relations, represented by energy and signal links, capture the interaction pathways between the components. Complex systems, such as a power distribution system, may operate in multiple configurations. The ability to change from one configuration to another is modeled by switching elements. Sets of components can also be grouped together to define subsystems. The modeling paradigm is implemented as part of the Fault Adaptive Control Technology (FACT) tool suite developed at Vanderbilt University (Manders *et al.* 2006). FACT employs the Generic Modeling Environment (GME) to present modelers with a component library organized as hierarchical collection of *components* and *subsystems* and a graphical interface for creating component-oriented system models (Ledeczi *et al.* 2001). The VIRTUAL ADAPT models created using FACT capture a number of possible ADAPT testbed configurations.

Each component includes an internal behavior model and an interface through which the component interacts with other components and the environment. The interfaces include two kinds of ports: (i) *energy ports* for energy exchange between the component and other components, and (ii) *signal ports* for input and output of signal values from the component. The current VIRTUAL ADAPT testbed component library includes models for the chargers, batteries, inverters, loads, relays, circuit breakers, and sensors that exist on the current ADAPT hardware testbed. For experimental purposes and "what if" analyses, new components can be added to the library by modifying existing component models or by creating new ones. System models are built by creating configurations of component models.

Many ADAPT testbed components are physical processes that exhibit hybrid behaviors, i.e., mixed continuous and discrete behaviors. These components are modeled as Hybrid Bond Graph (HBG) fragments (Mosterman and Biswas 1998). HBGs extend the bond graph modeling language (Karnopp, Margolis, and Rosenberg 2000) by introducing junctions that can switch *on* and *off*. Bond graphs are a domain-independent, topological modeling language based on the conservation of energy and continuity of power.

Building complete HBG models for a system requires detailed knowledge of the system configuration and component behaviors, as well as component parameters. This knowledge is typically obtained by consulting system designers and experts, extracting information from device manuals and research papers, and using experimental data collected during system operations. When experimental data is used, unknown parameters and functional relations associated with the models are estimated using system identification techniques. Often, this is a difficult task that requires significant analysis.

Model validation is performed by comparing simulated behaviors with data collected from experimental runs on ADAPT. A number of parameter estimation iterations may be necessary to obtain an accurate model of the system, keeping in mind the tradeoff between model accuracy and model complexity.

Generating Efficient Simulation Models

VIRTUAL ADAPT models created using the FACT tool suite can be translated into MATLAB Simulink® models using a systematic model transformation process (Roychoudhury *et al.* 2007) that is implemented using GME interpreters. The two-step process first transforms the HBG models into an intermediate block diagram representation, and then converts the block diagram representation into an executable Simulink model. The use of the intermediate block diagram provides the flexibility of generating executable models for other simulation environments with minimal effort.

Using naïve methods to generate executable hybrid system models requires pre-computation of the model for all possible system configurations or modes of operation. This is space-inefficient. The alternative is incremental generation of model structure at runtime when mode changes occur, which is time-inefficient (Daigle *et al.* 2006). We use a structurally static system model where the correct operating configuration of each bond graph element is selected online when mode changes occur, thus reducing excessive space and time costs at runtime. These algorithms exploit causality in bond graphs and, in addition to incremental generation, can also minimize the use of high-cost fixed point or algebraic loop solvers. Algebraic loop structures may arise in the Simulink structures to accommodate the switching functions in the HBG models.

In addition to generating nominal behaviors, the simulation system provides an interface through which sensor, actuator, and process faults with different "fault profiles" (e.g., abrupt versus incipient) can be injected into the system at specific time points. The Simulink model then generates faulty system behavior, which can form the basis for running diagnosis, prognosis, and fault-adaptive control experiments. In general, many different user roles and test articles, such as controllers, fault detectors, diagnosers, and interfaces for observing behaviors, can be tested.

Example: Battery Component Modeling

We demonstrate our modeling approach by developing a model of the batteries on the ADAPT testbed. The battery component model is developed from an electrical equivalent circuit model (Ceraolo 2000), shown in Figure 4 (left). The model computes the output battery voltage and the current flowing from the battery to a connected load when the battery is discharging; or from the charger to the battery, when it is charging. In both situations, some of this current goes into charging or discharging the batteries, and the rest is lost to parasitic reactions (e.g., gas production)

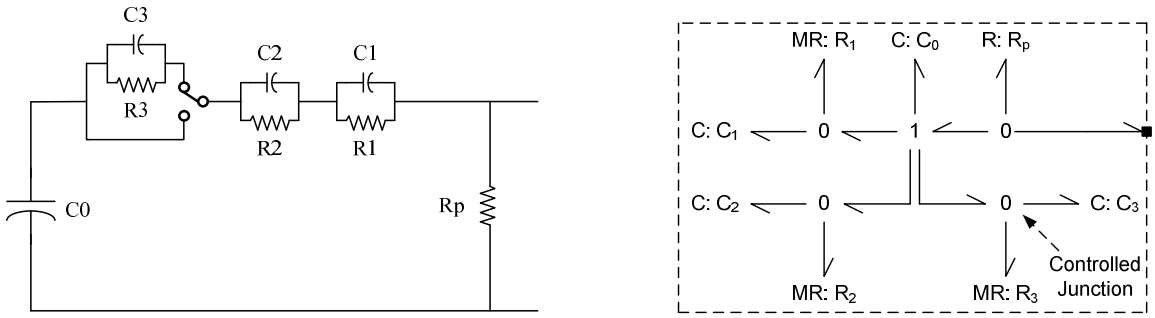


Figure 4: The equivalent circuit of the battery (left) and its corresponding hybrid bond graph (right).

modeled by a resistive element, R_p. The capacitor C_0, which has a large capacitance value, models the steady-state voltage of the battery. The steady-state voltage of the battery is a linear function of this capacitance value and the current amount of charge in the battery. The remaining resistor-capacitor pairs model the internal resistance and parasitic capacitance of the battery. All of the parameter values are nonlinear functions of system variables, such as state of charge and temperature. The nonlinear charging and discharging of the battery is captured as distinct modes of operation. Moreover, the internal battery model components differ for the different modes of operation. For example, the R_3-C_3 pair is only active during the charge mode. The two configurations are modeled by a switch in Figure 4 (left). Other configuration changes include switching between a load and a charger in the discharge versus charge modes.

Figure 4 (right) shows the HBG model of the equivalent circuit of the battery. The capacitors and resistors are in one-to-one correspondence with the electrical circuit elements. The hybrid nature of the battery is modeled by a controlled 0-junction, which is switched on and off depending on the direction of current through the battery. Most of the parameters in the battery HBG are nonlinear, and these nonlinearities are captured by making the bond graph element parameters nonlinear functions of system variables, such as the battery state of charge (SOC) and depth of charge (DOC). These two variables are computed in another portion of the model, not shown in Figure 4. As an example, the resistance of R_2 is proportional to the natural logarithm of the DOC. The variable parameters are indicated by prefixing their type-name with the letter "M", e.g., MR.

We have performed extensive system identification to estimate the parameters of the battery model, and have obtained good matches to actual observed behavior. Figure 5 shows a comparison of actual and simulated battery voltage in the battery discharge mode. The battery begins at a steady state, and as soon as a load is attached, it begins to discharge. As the battery nears its terminal voltage, the load is taken offline, and the battery begins to approach its new steady-state value. A battery charger is then connected, and the charging process generates an increase in the observed battery voltage.

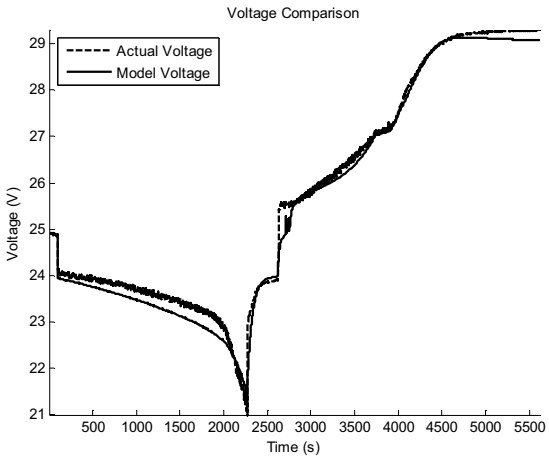


Figure 5: Comparison of actual and simulated battery voltages through discharge, no load, and charge modes.

Fault Injection

ADAPT supports the repeatable injection of faults into the system. Most of the fault injection is currently implemented via software using the *Antagonist* role described previously. Software fault injection includes one or more of the following: 1) sending commands to the testbed that were not initiated by the *User*; for this case the *Test Article* will not see the spurious command to the testbed, since it was not sent by the *User*; 2) blocking commands sent to the testbed by the *User*; 3) altering the testbed sensor data; for this case the *Test Article* and the *User* will see the altered sensor data since these reflect the faulted system. The sensor data can be altered in a number of ways, as illustrated in Figure 6. For a static fault, the data are frozen at previous values and remain fixed. An abrupt fault applies a constant offset to the true data value. An incipient fault applies an offset that starts at zero and grows linearly with time. Excess sensor noise is introduced by adding Gaussian or uniform noise to the measured value. Future work will add intermittent data faults, data spikes, and the ability to introduce more than one fault type for a given sensor at the same time. By using these three approaches to software fault injection, fault scenarios may be constructed that represent diverse component faults.

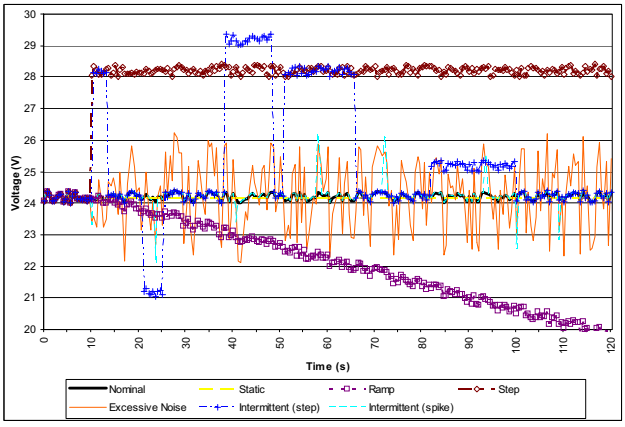


Figure 6: Example fault types.

In addition to the faults that are injected via software, faults may be physically injected at the testbed hardware. A simple example is tripping a circuit breaker using the manual throw bars. Another is using the power toggle switch to turn off the inverter. Relays may be failed by short-circuiting the appropriate relay terminals. Wires leading to or from sensors may be short-circuited or disconnected. Additional faults include blocking portions of the photovoltaic panel and loosening the wire connections in power-bus common blocks. Faults may also be introduced in the loads attached to the EPS. For example, one load is a pump that circulates fluid through a closed loop with a flow meter and valve. The valve can be closed slightly to

vary the back pressure on the pump and reduce the flow rate. Table 4 lists the faults that are injected into the testbed.

Since some fault scenarios may be costly, dangerous, or impossible to introduce in the actual hardware, VIRTUAL ADAPT also provides fault injection capabilities. For example, degradation in the batteries can be simulated as an incipient change in a battery capacitance parameter. Other parametric faults can also be injected and simulated. In addition, VIRTUAL ADAPT permits experimentation with fault scenarios that cannot be realized in the hardware, such as an inverter malfunction. Currently, mostly discrete failures (e.g., relay failures) and sensor errors are introduced into ADAPT, so the simulation provides added functionality by enabling injection of other types of fault scenarios.

Table 4: Faults injected into testbed.

Fault Description	Fault Type
Circuit breaker tripped	discrete
Relay failed open	discrete
Relay failed closed	discrete
Sensor shorted	discrete
Sensor open circuit	discrete
Sensor stuck	discrete
Sensor drift	continuous
Excessive sensor noise	continuous
AC inverter failed	discrete
PV panel blocked	continuous
Loose bus connections	continuous
Battery faults	continuous
Load faults	discrete, cont.

Test Articles

The test articles to be evaluated in ADAPT are health management applications from industry, academia, and government. The techniques employed may be data-driven, rule-based, model-based, or a combination of different approaches. Health management concepts to be investigated include fault detection, diagnosis, recovery, failure prediction and mitigation.

The technologies currently integrated with the testbed include model-based reasoning tools HyDE – Hybrid Diagnostic Engine (Narasimhan, Dearden, and Benazera 2004); FACT – Fault Adaptive Control Technology (Manders *et al.* 2006); and TEAMS-RT – Testability Engineering and Maintenance System Real Time (Deb *et al.* 1998). These tools are from government, academia, and industry, respectively. Each test article uses the ADAPT API to connect to the ADAPT message server, subscribing to the appropriate commands and data, and publishing the diagnosis results.

Each tool employs different abstraction, modeling, and reasoning methodologies. For example, TEAMS-RT typically discretizes continuous-valued sensor data into pass/fail test results. Cause-effect dependencies in a failure space, multi-signal model that link causes (components) to effects (test results) are used to isolate the fault. In contrast, FACT includes a hybrid observer that tracks continuous system behavior and mode changes. Statistical tests are used to detect a fault. Qualitative and quantitative fault signatures are used for fault isolation.

We aim to explore the advantages and disadvantages of the different approaches to health management to better understand their applicability to different fault types and operational contexts.

Performance Assessment

The purpose and scope of assessments and experiments as they relate to diagnostic and prognostic techniques and systems can vary significantly. We now identify a few different classes of assessments.

I. *Platform assessment:* Processors (CPUs) and operating systems may react differently to different diagnostic workloads. Therefore, it can be of interest to test an implementation on different computational platforms.

II. *Implementation assessment:* The programming language and the data structures can have a substantial impact on performance; hence it may be of interest to compare different implementations of the same algorithm.

III. *Algorithm assessment:* Different algorithms may solve the same computational problem. For instance, the problem of computing marginals in Bayesian networks can be solved using clique tree clustering or other approaches. Typically, this type of assessment involves the use of problem instances (Mengshoel, Wilkins, and Roth 2006).

IV. *Technique assessment:* The problem of electric power system diagnosis can be addressed using, for example, model-based reasoners, Bayesian networks or artificial neural networks.

V. *System assessment:* Overall system performance may also depend on the human(s) in the loop, and how they use and interact with different automated diagnostics systems.

Initial efforts will focus on classes III, IV, and V. The assessment of different diagnostic algorithms will consist of metrics determined from the compilation of several test runs, which include nominal and faulty behavior. For a single test run, it will be classified as a false alarm if a fault was not injected during the run but the test article reported one or if the test article reported a fault before it was injected. If the test article does not report a fault when one was injected, it will be classified as a missed alarm. The correctness and precision of fault isolation will also be determined for each run. By combining the results over several runs, false alarm rates, missed alarm rates, and isolation rates for the test article will be measured. Additional metrics include fault detection and isolation times. Not all of the metrics will apply to each test article. For example, a particular technique may only perform fault detection without isolating the cause of the fault. Additional studies will include variations in the amount and variability of data available to the test articles, i.e., only data from voltage sensors is available or some data is available intermittently.

Conclusions

This paper describes a testbed at NASA Ames Research Center for evaluating and maturing diagnostic and prognostic concepts and technologies. The electrical power system hardware, together with the software architecture and unique concept of operations, offers many challenges to diagnostic applications such as a multitude of system modes, transient behavior after switching actions, multiple faults, and load-dependent noise. A simulation model is available to facilitate the development and testing of health management applications. Future work will include the integration of more test articles, loads, faults, operational scenarios, and evaluation techniques. In meeting the five goals of the testbed listed in the Introduction, much greater knowledge of the trade-offs among diagnostic technologies will be available to system designers for future programs. Many current technology assessments have relied upon trade literature or technical publications of an algorithm's definition and performance. ADAPT provides a technology-neutral basis for the comparison of various techniques and a highly configurable operational environment for the evaluation of an algorithm's performance under specific operational requirements and fault conditions.

Acknowledgments

The authors thank Somnath Deb, Sudipto Ghoshal, Charles Domagala and Venkat Malepati from Qualtech Systems, Inc. for creating the TEAMS model of ADAPT under NASA contract number NNA06AA51Z and the TEAMS-RT application for the testbed under NASA contract number NNA06AA64C, and for their many discussions on the design of the diagnostic experiments. Contributions by RIACS were based upon work supported by NASA under award NCC2-1426. Contributions by Perot Systems were supported by NASA under contract NNA04AA18B and contributions by SAIC were supported by NASA under contract NAS2-02091. Contributions by Vanderbilt University were supported by NASA USRA grant 08020-013, NASA NRA grant NNX07AD12A, and NSF CNS-0615214.

References

Williams, B. C.; Nayak, P.; and Muscettola, N. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1–2):5–48.

Mosterman, P. J.; and Biswas, G. 1998. A Theory of Discontinuities in Physical System Models. In *J Franklin Institute* 335 B(3):401–439.

Karnopp, D. C.; Margolis, D. L.; and Rosenberg, R. C. 2000. *Systems Dynamics: Modeling and Simulation of Mechatronic Systems.* New York: John Wiley & Sons, Inc., 3rd edition.

Manders, E. J.; Biswas, G.; Mahadevan, N.; and Karsai, G. 2006. Component-Oriented Modeling of Hybrid Dynamic Systems Using the Generic Modeling Environment. In *Proceedings of the 4th Workshop on Model-Based Development of Computer Based Systems*.

Narasimhan, S., and Biswas, G. 2007. Model-Based Diagnosis of Hybrid Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, Part A, 37(3):348–361.

Roychoudhury, I.; Daigle, M.; Biswas, G.; Koutsoukos, X.; and Mosterman, P. J. 2007. A Method for Efficient Simulation of Hybrid Bond Graphs. In *Proceedings of the International Conference on Bond Graph Modeling (ICBGM 2007)*, 177–184.

Daigle, M.; Roychoudhury, I.; Biswas, G.; and Koutsoukos, X. 2006. Efficient Simulation of Component-Based Hybrid Models Represented as Hybrid Bond Graphs. Technical Report ISIS-06-712, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA.

Ceraolo, M. 2000. New Dynamical Models of Lead-Acid Batteries. *IEEE Transactions on Power Systems* 15(4):1184–1190.

Mosterman, P. J., and Biswas, G. 1999. Diagnosis of Continuous Valued Systems in Transient Operating Regions. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 29(6):554–565.

Mengshoel, O. J.; Wilkins, D. C.; and Roth, D. 2006. Controlled Generation of Hard and Easy Bayesian Networks: Impact on Maximal Clique Tree in Tree Clustering. *Artificial Intelligence*, 170(16–17):1137–1174.

Ledeczi, A.; Maroti, M.; Bakay, A.; Nordstrom, G.; Garrett, J.; Thomason IV, C.; Sprinkle J.; and Volgyesi P. 2001. GME 2000 Users Manual (v2.0).

Narasimhan, S.; Dearden, R.; and Benazera, E. 2004. Combining Particle Filters and Consistency-Based Approaches for Monitoring and Diagnosis of Stochastic Hybrid Systems. *15th International Workshop on Principles of Diagnosis (DX04)*, Carcassonne, France.

Deb, S.; Mathur, A; Willitt, P; and Pattipati, K. 1998. Decentralized Real-Time Monitoring and Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics.*

Analyzing the influence of temporal constraints in possible conflicts calculation for model-based diagnosis

Belarmino Pulido† and **Carlos Alonso**† and **Anibal Bregón**† and **Vicenç Puig**‡ and **Teresa Escobet**‡
† Dpto. de Informática. Universidad de Valladolid. Valladolid (Spain)
e-mail: {belar,calonso,anibal}@infor.uva.es
‡ Dept. ESAII, Universitat Politècnica de Catalunya, Tarrasa (Spain),
e-mail: {vicenc.puig,teresa.escobet}@upc.edu

Abstract

Diagnosis for dynamic systems is still an open research issue in the model-based diagnosis community. DX and FDI communities have traditionally approached the problem in very different, but complementary ways. Recently, the work on BRIDGE has provided a common framework for analyzing results from both communities for static systems. In this work, we propose a first step towards the integration of temporal information within the BRIDGE framework. We analyze the influence of dynamic constraints selection in the behavior estimation capabilities for dependency-compilation techniques from DX and FDI worlds.

Main results are that both of them have the ability to generate ARR-like expressions which can be implemented as simulators or predictors. Additionally, algorithms computing Possible Conflicts provide the implicit structural model for state-observer design with no extra knowledge in the model.

Keywords: Consistency-based Diagnosis, Dynamic Systems, BRIDGE, Structural Analysis

Introduction

FDI and DX approaches to model-based diagnosis have worked on diagnosing dynamic systems using different kinds of models, different assumptions concerning robustness issues (w.r.t. disturbances, modelling errors or noise), and so on.

Recently, the BRIDGE community (Biswas *et al.* 2004), based on the work by Cordier et al. (Cordier *et al.* 2004) have provided a common framework for sharing results and techniques, at least for static systems. The basis for the work is the comparison of two model-based diagnosis techniques: consistency-based diagnosis via conflicts (Reiter 1987), and fault detection and isolation via Analytical Redundancy Relations, ARRs for short, obtained through structural analysis (Staroswiecki 2002; Blanke *et al.* 2003).

This work tries to follow one of the research issues suggested by Cordier et al. (Cordier *et al.* 2004): analyzing the influence of temporal information in the BRIDGE framework. As a first step, we analyze the influence of temporal information in two different dependency-compilation techniques: Possible Conflict calculation, which follows a DX perspective, and ARRs obtained through structural analysis, which follows a FDI perspective.

Recently, it was demonstrated the strong similarity between possible conflicts, or PCs, and ARRs for static systems (Pulido & Alonso-González 2004) within the BRIDGE framework. This work is built upon those results and extend the comparison to dynamic systems. The final goal of this work is to establish clear links between FDI and DX concerning temporal issues in order to ease the integration of techniques from both fields in a given diagnosis system. For instance, different architectures have been proposed that integrates FDI and DX techniques (Mosterman 1997). Our initial guess is that strong similarities between compilation techniques from DX and FDI approaches can be helpful in the integration process.

The organization of this paper is as follows. Next section summarizes the fundamental approaches to model-based diagnosis of dynamic systems in both communities. Afterwards, a summary of Model-based diagnosis using PCs and ARRs is introduced. Next section provides a case study to facilitate the understanding of the comparison in the following section. Later on, the inclusion of temporal information in both approaches is compared. Finally, we show some results on the case study, and draw some conclusions.

Model-based diagnosis of dynamic systems: DX and FDI classical approaches

The DX approach has a solid theoretical ground for static systems (Reiter 1987; de Kleer, Mackworth, & Reiter 1992). Consistency-based diagnosis, or CBD, is its main theoretical framework and GDE (de Kleer & Williams 1987), its computational paradigm. However, there is no general framework for consistency-based diagnosis of dynamic systems (de Kleer 2003; Travé-Massuyès & Dague 2003). In fact, several works have demonstrated the wide variety of definitions and methods used by the DX community (Brusoni *et al.* 1996).

Main concern in DX has been the localization and identification stage in the diagnosis process. The presence of uncertainties in the model or disturbances has been solved by means of qualitative or semi-qualitative models (being qualitative modelling a key research area in the field). However, the coupling of qualitative models and GDE-like diagnosis

systems has given a wide variety of problems:

- The presence of temporal information in the qualitative models – i.e. qualitative differential equations– brought new difficulties to behavior estimation. If this process is achieved through constraint propagation, propagation through temporal constraints, back and forward, implies integration or differentiation processes. The work by Chantler et al. (Chantler, Daus, & Coghill 1996) demonstrated that both approaches could be equivalent. Nevertheless, most researchers in DX have opted for some kind of simulation based on integration, using only quantitative or qualitative integration methods (Bouamama 2000; Mosterman, Manders, & Biswas 2000). An exception is State-based diagnosis (Struss 1997), which is based on qualitative deviations of variables, and does not perform the integration step, assuming the qualitative values of the derivatives are provided.
- On-line dependency-recording and qualitative modelling can lead to a feedback loop problem (Dressler 1996). Although different proposals were made in the past, to skip this problem two main paths are followed:
  - The use of forward propagation through the constraints to predict the behavior. Afterwards, and only if a discrepancy is found, there is a backward propagation step to find out the dependencies. This task is usually done by searching in a causal or functional structure, as in CAEN (Travé-Maussuyès, Escobet, & Quevedo 2000), DYNAMIS (Chittaro, Ranon, & Lopez Cortes 1996) or TRASCEND (Mosterman 1997).
  - Off-line dependency-compilation, following similar ideas to the structural analysis of the FDI community, computes off-line the set of components or subsystems able to become conflicts. Afterwards, in the on-line stage, these subsystems estimate the behavior of variables searching for a discrepancy (Loiez & Taillibert 1997; Washio, Sakuma, & Kitamura 1997; Pulido & Alonso-González 2004).

The FDI approach, using mainly numerical analytical models, has a more standard specification of dynamic aspects (Blanke *et al.* 2003; Travé-Massuyès & Dague 2003). The mathematical model can be expressed in a variety of ways: state-space models, input-output models, or even black-box models obtained through identification. Main focus on the FDI approach is fault detection, generating a set of fault indicators, called residuals, which should be activated in presence of faults (a similar concept to conflict detection in DX). Residual expressions come from analytical redundancy in the system. Once residuals are activated, some kind of decision logic is used to provide fault localization or isolation. In the FDI community, there are solid theoretical results for linear systems (Gertler 1998; Patton, Frank, & Clark 2000; Blanke *et al.* 2003), while analysis of non-linear systems is a major research issue.

A big concern in FDI research is robustness in the detection stage regarding disturbances, modeling uncertainties, and measurements noise. These problems affect the residual evaluation stage: residuals may become non-zero even in non-faulty situations. A great research effort has been made to accomplish the decoupling of residuals from these errors (Gertler 1998; Patton, Frank, & Clark 2000).

Residual generation techniques can be split in two main categories: state-observers, and parameter estimation (Gertler 1998; Travé-Massuyès & Dague 2003). First one uses system measurements to estimate the state variables of the system, looking for discrepancies. Such approach can be tackled via parity space, or observer techniques. Second one identifies system parameters, which should remain constant. This approach is based on parameter identification (Isermann 1993). It has been demonstrated that those techniques can be considered as equivalent as far as linear models are concerned (Gertler 1998).

Regarding the fault isolation stage, major research effort is made in achieving residuals which are sensitive to a set of faults and insensitive to others. The design of structured or directional residuals are the two major trends in the field (Gertler 1998), paying special attention to robustness against disturbances or modeling errors (Travé-Massuyès & Dague 2003).

Possible Conflicts vs Analytical Redundancy Relation

Possible conflicts: a dependency-compilation technique for consistency-based diagnosis

Possible conflicts, PCs for short, are those sub-systems capable to become conflicts in CBD, i.e. *minimal subsets of equations containing enough analytical redundancy to perform fault diagnosis*[1] (Pulido & Alonso 2000). In GDE the set of conflicts are computed on-line (de Kleer & Williams 1987). However, many approaches have opted recently for the off-line computation of conflict-like structures: they are called *dependency-compilation techniques*.

The main idea behind the *possible conflict* concept is that the set of subsystems capable to generate a conflict can be calculated off-line, through the analysis of the set of equations in the original model. The PCs computation process is done in three steps.

The first one generates an abstract representation of the system, as an hypergraph. In this representation there is just qualitative information about constraints in the models, and their relationship to known and unknown variables in such models.

The second step looks for minimal over-determined sets of relations, which are essential for model-based diagnosis. These subsystems, called *minimal evaluation chains*, or MECs, represent a necessary condition for a conflict to exist. Each minimal evaluation chain, which is a partial subhypergraph of the original system description, need to be solved using only local propagation criteria (to follow GDE computational framework).

In the third step, extra knowledge is added to fulfill that requirement. Each possible way a constraint can be solved,

[1] The possible conflict represent, in FDI terms, an ARR which could be used for fault detection and isolation purposes.

by means of local propagation, is specified[2]. As a consequence, each minimal evaluation chain generates a directed and-or graph. In each and-or graph, we search for every possible way the system can be solved using local propagation. Each possible way is called a *minimal evaluation model*, or MEM, and it can be used to predict the behavior of a subsystem. Moreover, since conflicts will arise only when models are evaluated with available observations, the set of constraints in a minimal evaluation model is called a *possible conflict*, PC for short.

Each MEM describes an executable model, which can be used to perform fault detection. If there is a discrepancy between predictions from those models and current observations, the possible conflict could be responsible for such a discrepancy and should be confirmed as a real conflict. Afterwards, diagnosis candidates are obtained from conflicts following Reiter's theory.

PCs calculation use minimality criteria in terms of sets of constraints. Nevertheless, it is straightforward to obtain candidates based on components.

As pointed out in (Pulido & Alonso-González 2004), the set of MEMs generated with this approach is equivalent to the set of conflicts computed by the GDE.

Model-based FDI using ARRs

In the FDI approach, residuals can be obtained through analytical redundancy in the models. One way to obtain the set of residuals is to compute the set of Analytical Redundancy Relations, or ARRs. A brief summary of ARR computation by means of structural analysis is provided. Further details can be found in (Blanke *et al.* 2003).

The set of over-determined systems for diagnosis is obtained from the unique canonical decomposition of the structural description of the system to be diagnosed into under-determined, just-determined, and over-determined sets of constraints. The canonical decomposition is based on finding a complete matching, w.r.t. unknown variables, in the bipartite graph associated to the structural description of the system. Combination of just-determined systems together with redundant relations is the basis for the set of ARRs.

Each complete matching in the bipartite graph can be considered as a causality assignment, but it is necessary to obtain a real causal matching for the over-determined system, from the set of causal assignments satisfying the invertibility condition[3]. Each ARR can be solved afterwards using observations, providing a residual, and the residual can be used for diagnosis purposes.

It should be noticed that all the steps, except for residual computation, are done off-line. Hence, computing ARRs is a compilation technique in FDI.

[2]Each hyper-arc in the system description has one or more related and-or arcs, representing possible causal assignments to solve the constraint.

[3]These are the interpretations for constraints in the possible conflict approach

A case study

The laboratory plant shown in figure 1 will be used to illustrate the concepts concerning the comparison of both techniques. The plant resembles common features of industrial continuous processes. It is made up of four tanks $\{T_1, \ldots, T_4\}$, five pumps $\{P_1, \ldots, P_5\}$, and two PID controllers acting on pumps P_1, P_5 to keep the level of $\{T_1, T_4\}$ close to the specified set point. To control temperature on tanks $\{T_2, T_3\}$ we use two resistors $\{R_2, R_3\}$, respectively.

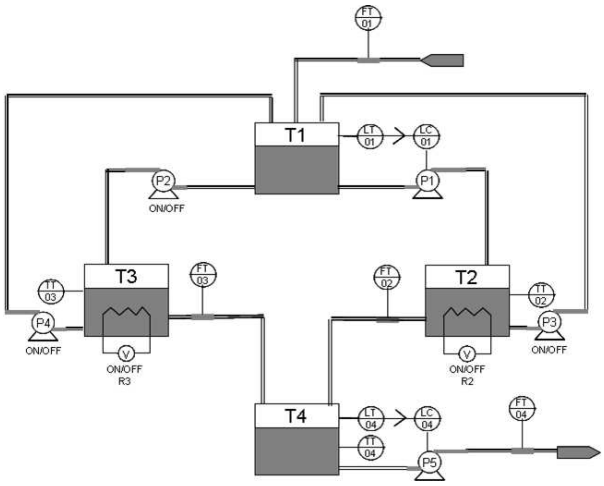


Figure 1: Diagram of the laboratory plant.

In this plant we have eleven different measurements: levels of tanks T_1 and T_4 –$\{LT01, LT04\}$–, the value of the PID controllers on pumps $\{P_1, P_5\}$ –$\{LC01, LC04\}$–, inflow on tank T_1 –$\{FT01\}$–, outflow on tanks $\{T_2, T_3, T_4\}$ –$\{FT02,\ FT03,\ FT04\}$–, and temperatures on tanks $\{T_2, T_3, T_4\}$ –$\{TT02,\ TT03,\ TT04\}$–. Action on pumps $\{P_2, P_3,\ P_4\}$, and resistors –$\{R_2, R_3\}$– are also known.

This plant can work on different situations, commanded through different operation modes. In the selected operation protocol: resistor R_3 is switched off, while resistor R_2 is on. Also, pumps $\{P_3, P_4\}$ are switched off; hence, just flow $FT01$ is incoming to tank T_1.

The behavior of the plant has been modelled using classical equations based on first principles:

t_{dm}	mass balance in tank t
t_{dE}	energy balance in tank t
t_{fb}	flow from tank t to pump
t_f	flow from tank t through a pipe
r_p	resistor failure

The reader can find a complete definition of the set of faults considered and possible conflicts found in the plant in (Bregón *et al.* 2006).

Introducing temporal information

Temporal issues in PCs calculation

Two kinds of constraints are used to model dynamic behavior in the possible conflict approach (Pulido, Alonso, & Acebes 2001):

- *Instantaneous* or *algebraic* constraints, which model static behavior and are graphically represented as solid edges in the hypergraph,
- *Differential* constraints, which model dynamic behavior (by means of differential equations). Only the differential constraint $(v_i, \frac{dv_i}{dt})$ is allowed in the system description. Differential constraints are graphically represented as dashed edges in the hyper-graph.

Using this canonical notation, we do not restrict the order of the derivatives to be used[4].

In order to compute Minimal Evaluable Models, we need to introduce interpretations for differential constraints. In the related and-or graph, each differential constraint may have two different interpretations:

$$v_i \dashrightarrow \frac{dv_i}{dt} \quad or \quad \frac{dv_i}{dt} \dashrightarrow v_i$$

Propagation back and forward through differential arcs implies using integration or differentiation in the executable models[5]. Hence, the meaning for these interpretations is:

$$\frac{dv_i}{dt} \dashrightarrow v_i \quad v_i(t+h) = \int_t^{t+h} \frac{dv_i(t)}{dt} + v_i(t) \tag{1}$$

$$v_i \dashrightarrow \frac{dv_i}{dt} \qquad \frac{dv_i(t)}{dt} = \frac{v_i(t+h)-v_i(t)}{h} \tag{2}$$

Similar ideas have been studied by DX researchers working with qualitative models containing derivatives: intra-state constraints for static relations, and inter-state constraints for dynamic relations (Dressler 1996). Propagation through inter-state constraints, as in equation 1, implies integration, and this is the basis for simulation based approaches. On the other hand, derivative methods rely on propagation through equation 2.

Different authors from both DX and FDI (Chantler, Daus, & Coghill 1996; Staroswiecki 2002) have demonstrated that integration and derivative methods can be equivalent regarding behavior estimation, assuming adequate sampling rates and precise approximations were available, or assuming initial conditions are known.

Analyzing cyclical configurations

Off-line dependency-recording using Possible Conflicts avoids the feedback loop problem in qualitative simulation (Dressler 1996), providing the whole set of systems capable to become conflicts in the on-line stage.

Moreover, to produce GDE-like results, the local propagation criterion must be taken into account while computing the set of PCs. This criterion allows GDE to trace back those correctness assumptions involved in a conflict. As pointed out by different authors (Katsillis & Chantler 1997) local propagation can be easily halted. Cyclical structures found in Minimal Evaluable Models could be responsible for halting local propagation. Different cyclical configurations can be found:

- **Loops containing just instantaneous constraints**.
  These loops present two different configurations:
  - One of the magnitudes in the loop can be observed or directly estimated from observations. In this case, there is no loop, and one magnitude will be a discrepancy node.
  - The loop can not be broken using observations. It describes an algebraic loop. Such loops go against the locality principle, and can be considered as super-components or super-constraints for fault localization purposes (every constraint in the loop is responsible for the estimation (Katsillis & Chantler 1997)).
- **Loops containing both instantaneous and differential constraints**.
  In this case:
  - If an integration approach is used, there is no such a loop. Each magnitude in the loop has an associated temporal index, hence prediction in time point t is clearly distinct from prediction in time point $t+1$.
    In fact, this configuration is graphically described as an spiral. This issue has been previously studied for qualitative models by Dressler et al. (Dressler 1996).
  - In a derivative approach, no loop containing the magnitude and its derivative is allowed, because it could not be instantaneously solved. If the state variable can be measured, the derivative can be solved if an estimator is used.

All these possible cyclical configurations included in a MEM can be easily detected and analyzed off-line, while the set of MEMs is computed (Pulido & Alonso 2001). New developed algorithms for finding cycles in the MEM can be found in he appendix.

We will illustrate these concepts graphically in our case study. The plant in figure 1 provides several examples, because there are equations modelling mass or energy balances in tanks, i.e. differential constraints. For instance, the subsystem made up of $\{T_1, P_1, T_2\}$ provides a Possible Conflict, PC_2, whose MEM can be seen in figure 2. Outflow from T_2, $FT02$, is simulated. Based on the actual level on T_1, $LT01$, and the PI controller, PI_01_u, inflow to T_2, $LF2\_F$, is estimated. Afterwards, mass of tank T_2, $T2\_M$ can be computed. This magnitude, is a state variable, which can be used to compute the height on T_2, $T2\_H$, and the estimated outflow $LF4\_F$. In this PC, the initial condition $T2\_M(0)$, or equivalently $T2\_H(0)$, must be known for simulation purposes. In this case, there is no loop in the path: $T2\_H(t), LF4\_F(t), T2\_H'(t), T2\_H(t+1)$. Magnitudes have different time indices; hence, it is an spiral representing simulation.

Temporal issues in ARR calculation

In the structural analysis for Analytical Redundancy Relations computation, temporal information is introduced in the model through the differential constraint concept. The inclusion of causal matchings in those constraints gives rise to integral/derivative interpretations.

[4]For instance, second order derivatives can be introduced by means of a differential constraint $(\frac{dv_i}{dt}, \frac{d^2v_i}{dt^2})$, and so on.

[5]These concepts are equivalent to causal matchings for differential constraints in the structural approach for ARRs.

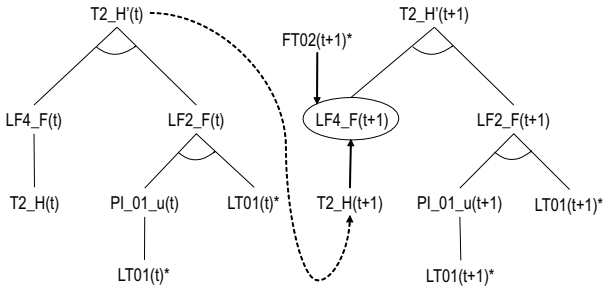


Figure 2: Possible conflict PC_2: one MEM related to PC_2 provides the simulation of flow $LF4\_F$, which can be compared against measured $FT02$.

Integral matchings will provide ARRs where the relation between the state variable $x(t)$ and its derivative $\dot{x}(t)$ represent:

$$x(t+h) = \int_t^{t+h} \dot{x}(t) \cdot dt + x(t)$$

As pointed out by Staroswiecki (Blanke *et al.* 2003), this is a simulation approach which needs to know the value for $x(0)$ (initial conditions must be known for simulation, or they should be directly calculated from observed variables).

On the other hand, derivative matchings give ARRs where variable $x(t)$ and its derivative $\dot{x}(t)$ represent:

$$\dot{x}(t) = \frac{dx(t)}{dt}$$

This approach provides an estimation of the variable in the next time step using the computation of its derivative at the current time. This fact, of course, gives rise to the well-known problem of derivative estimation from available measurements: sensitivity to noise, disturbances, and so on.

In this context, ARRs containing cyclical structures are also analyzed (Staroswiecki 2002):

- for static systems they identify algebraic loops which can not be solved using local propagation.
- for dynamic systems: integral loops are allowed, but differential loops are not allowed.

Comparing temporal issues in PC and ARR calculation

At it has been shown, it is quite clear that assumptions concerning temporal constraints are identical for computing Possible Conflicts or ARRs. Main difference comes in practice.

Most DX works have traditionally opted by the simulation approach, because the derivative estimation is a process "fraught with difficulty when real signals are used" (Chantler, Daus, & Coghill 1996). On the other hand, in FDI differentiation is preferred, focusing on estimation or observation schemes (Blanke *et al.* 2003). Hence, FDI approaches will be concerned with numerical problems related to derivative estimation for real noisy measurements.

Concerning PCs calculation, they have been used for a semi-closed loop simulation approach (Pulido 2001), but PCs using derivatives can be easily obtained using just a different model description allowing derivative interpretations.

The derivative approach is preferred for ARR computation (Staroswiecki 2002; Blanke *et al.* 2003). Again, it seems obvious that ARRs containing integral causality could just be derived using the same algorithms, just introducing models allowing integral matchings.

As a consequence, our point is that using compilation techniques, both approaches can profit from each other, given their strong similarities.

As mentioned above, Chantler et al. (Chantler, Daus, & Coghill 1996) have demonstrated the equivalence of their estimation capabilities. Moreover, in the FDI approach, there is evidence of the equivalence of the three main techniques –simulation, estimation, and state observers– for linear systems (Gertler 1998). The general system description could be also seen as:

$$\dot{\hat{X}}(t) = A \cdot \hat{X}(t) + B \cdot U(t) + K \cdot (Y(t) - \hat{Y}(t)) \quad (3)$$

$$\hat{Y}(t) = C \cdot \hat{X}(t) \quad (4)$$

Depending on the selected value for the gain K, we obtain: from $K = 0$ for simulation, to $A = K \cdot C$ for prediction (Puig *et al.* 2006).

Moreover, in the case of non-linear models, there is no guarantee for equivalence among results coming from every MEM associated to a PC. The same can be said about ARRs associated to the same support-set (Pulido & Alonso-González 2004). In that case, evaluations of different MEMs or ARRs can provide different results for fault detection purposes.

Nevertheless, we think that the relevant issue is that both techniques still provide similar structural descriptions of subsystems –PCs or ARRS– which can be used for Fault Detection and Isolation purposes, using simulation or estimation approaches, in the BRIDGE context.

Finally, there is still another important issue. Given the equivalence of the three FDI approaches, what can be said about the relationship between state-observers and compilation techniques from a structural point of view?

Dependency-compilation and state-observer design

It was previously mentioned that the algorithm for computing possible conflicts provides a structural expression for the executable model related to a MEM. The executable model can be implemented as a simulator or an estimator. But, what about state-observers?

State observers work with a system description described by equations 3 and 4. It is used to minimize the error, $e(t) = y(t) - \hat{y}(t)$ -regarding some criteria- on the state estimation, using the set of available measurements. This step is independent of the type of observer -i.e. Luenberger, Extended Kalman Filter, etc.-, and the observer gain, which should be selected according to the desired expected behavior for the observer.

The set of PCs is computed following the GDE approach for dependency-compilation (Pulido & Alonso-González 2004). Hence, each PC has at least one MEM which has

exactly one discrepancy node in its associated and-or graph. Such node can be found as:

- an estimation for a measured magnitude,
- a double estimation of a non-measured magnitude.

When differential constraints are included, our algorithms provide an interesting side-result. When integral causality is used, the Minimal Evaluable Model can be implemented as a simulator or as a state-observer.

Proposition: *Those MEMs containing a state variable, can provide the minimal computational input-output form for an state observer, if there exists a path, from the observed variable to the estimated state variable, which contains no differential arcs.*

This result comes from the way the error $e(t)$ is introduced in a generalized state-observer scheme in equation 3, which is implicit in the MEM:

$$e(t) = Y(t) - \hat{Y}(t) = Y(t) - C \cdot \hat{X}(t)$$

$C \cdot \hat{X}(t)$ represents a direct estimation for an observed variable given a state-variable.

In our MEM, the path with no differential arcs can directly estimate an observed variable from the state variable. The path containing differential constraints provides an estimation of the current state from the previous one.

Since MEMs are minimal, that expression must describe a discrepancy (which would be used in a state-observer to correct the estimation of the state variable in the next time step).

Our algorithms are capable to provide every estimation for a state variable (using derivative, integration or algebraic equations). Hence, we can trace backward if there is a path in the and-or graph from the discrepancy node ($e(t)$) to the state variable which contains no differential and-or arc.

No additional information is required in the model description to provide these results. We just need to include one step at the end of the algorithms used for analyzing cyclical structures in the Minimal Evaluable Models (and-or graphs) related to a Possible Conflict.

We understand that there are two different stages in state-observers design:

1. Finding the structure (set of equations involved) of the observer, which faults can be associated to the observer, and so on.
2. Analyzing those equations for convergence, and robustness issues: what is the value of K, how the residual is evaluated, and so on.

Our work is just focused on the first step, providing the collection of minimal expressions for state-observers according to equation 3.

Results on the case study

The set of possible conflicts shown in table 1 has been found in the case study using just integral causality.

	$Constraints$	$Components$	$Estimate$
PC_1	$t1_{dm}, t1_{fb1}, t1_{fb2}$	T_1, P_1, P_2	$LT01$
PC_2	$t1_{fb1}, t2_{dm}, t2_f$	T_1, T_2, P_1	$FT02$
PC_3	$t1_{fb1}, t2_{dm}, r2_p$	T_1, P_1, T_2, R_2	$TT02$
PC_4	$t1_{fb2}, t3_{dm}, t3_f$	T_1, P_2, T_3	$FT03$
PC_5	$t1_{fb2}, t3_{dm}$	T_1, P_2, T_3	$TT03$
PC_6	$t4_{dm}$	T_4	$LT04$
PC_7	$t4_{fb}$	T_4, P_5	$FT04$

Table 1: Possible conflicts found for the laboratory plant; constraints, components, and the estimated variable for each possible conflict.

PCs in the plant were obtained using a simulation approach. If an estimation approach is used, i.e. just using differentiation, the set of possible conflicts found is smaller. We have found five possible conflicts, providing different fault isolation capabilities. In fact, only three of them are identical in terms of constraints involved to $\{PC_1, PC_2, PC_5\}$.

Concerning the design of executable models for a given MEM, let's take PC_2 as an example:

- PC_2 can be modeled as a simulator using the expression given by its MEM in figure 2.
- Or can be modeled as an state-observer, if the state variable $T2\_M$ –equivalent to $T2\_H$– is selected for the observer, as can be seen in figure 3.

  In this case, $T2\_M$ can be computed given previous values, and can also be corrected using the difference between the direct estimation of the outflow, $LF4\_F$, and the observed variable $FT02$: $e_{T2\_H}(t)$, being K the implicit error correction.

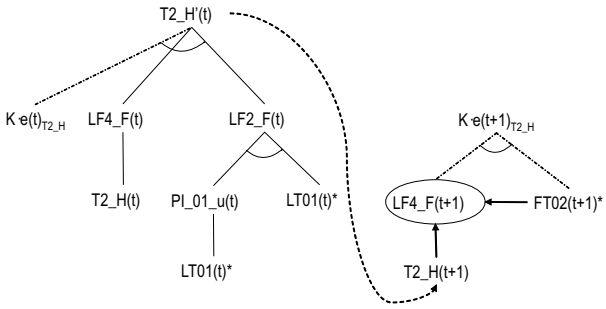


Figure 3: Possible conflict PC_2. Its original MEM can be interpreted as an state observer for $T2\_H$.

Conclusions

PCs and ARRs provide almost identical approaches to perform fault detection of static systems using compiled sets of analytical redundancy relations. Both approaches can find, off-line, the set of constraints needed for conflict detection or residual evaluation purposes.

The inclusion of dynamic information in the models provides different approaches to behavior estimation via prediction, simulation, or state-observers. Again, both approaches

are able to handle cyclical configurations, providing similar results. But it should be notice that using integral or derivative approaches can provide different sets of PCs, and consequently different fault detection and isolation capabilities.

These results are consistent with ongoing work in the FDI field to automatically analyze the influence of differential constraints within structural analysis, allowing different representation for differential arcs (Düstegör *et al.* 2006; Krysander 2006). Since these works use a different modelling approach, and different algorithms, it is necessary further research to provide a precise comparison with them.

The advantage of PC computation is the ability to provide not only models for simulation, but the implicit expression needed for state-observers design. Using just integral interpretations, our algorithms provide minimal sets of constraints needed for the design of state-observers. In this sense, it is not necessary to include additional equations for a given observer as in the ARR approach (Staroswiecki 2002). Of course, the proper values of the observer gain must be computed afterwards following the traditional criteria in FDI.

There are other results on the comparison of ARRs versus High-gain observers (Christophe, Cocquempot, & Jiang 2004). Those works are based on the analytical generation of the state-observers. Our approach must be considered as a previous step for state-observers design. Once the symbolic model is known, it can be analyzed to fulfill convergence, robustness or sensitivity criteria.

Main conclusion from this comparison is that dependency-compilation techniques from DX and FDI fields have approached diagnosis of dynamic systems almost identically in terms of simulation or prediction techniques. Differences come from the kind of models used to cope with uncertainty, noise, and disturbances.

The main task ahead is still to provide a framework for the integration of techniques from both fields for dynamic systems, supporting previous attempts to combine numerical models and structural methods (Mosterman 1997; Pulido, Alonso, & Acebes 2001).

Acknowledgments: This work was supported by the Spanish Ministry of Education and Culture (MEC 2005-08498).

References

Biswas, G.; Cordier, M.; Lunze, J.; Travé-Massuès, L.; and Staroswiecki, M. 2004. Diagnosis of complex systems: bridging the methodologies of the fdi and dx communities. *"IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics"* 34(5):2159–2162.

Blanke, M.; Kinnaert, M.; Lunze, J.; and Staroswiecki, M. 2003. *Diagnosis and Fault Tolerant Control*. Springer.

Bouamama, B. 2000. Qualitative modelling using bond graph models. In *Vacation School on Qualitative Methods for Fault Diagnosis*.

Bregón, A.; Pulido, B.; Simón, A.; Moro, I.; Prieto, O.; Rodríguez, J.; and Alonso-González, C. 2006. Focusing fault localization in model-based diagnosis with case-based reasoning. In *XVII Intl. WS. on Principles of Diagnosis, DX'2006*.

Brusoni, V.; Console, L.; Terenziani, P.; and Dupre, D. 1996. A spectrum of definitions for temporal model-based diagnosis. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, 44–52.

Chantler, M.; Daus, S. Vikatos, T.; and Coghill, G. 1996. The use of quantitative dynamic models and dependency recording engines. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, 59–68.

Chittaro, L.; Ranon, R.; and Lopez Cortes, J. 1996. Ship over troubled waters: Functional reasoning with influences applied to the diagnosis of a marine technical system. In *Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX-96)*, 69–78.

Christophe, C.; Cocquempot, V.; and Jiang, B. 2004. Link between high-gain observer-based and parity space residuals for fdi. *Trans. on the Institute of Measurement and Control* 26(325).

Cordier, M.; Dague, P.; Lévy, F.; Montmain, J.; Staroswiecki, M.; and Travé-Massuyès, L. 2004. "conflicts versus analytical redundancy relations: a comparative analysis of the model-based diagnosis approach from the artificial intelligence and automatic control perspectives". *"IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics"* 34(5):2163–2177.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32:97–130.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnosis and systems. In *Readings in Model Based Diagnosis*. Morgan-Kauffman Pub., San Mateo. 54–65.

de Kleer, J. 2003. Fundamentals of model-based diagnosis. Proceedings of 14th International Workshop on Principles of Diagnosis, DX'2003.

Dressler, O. 1996. On-line diagnosis and monitoring of dynamic systems based on qualitative models and dependency-recording diagnosis engines. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*, 461–465.

Düstegör, D.; Frisk, E.; Cocquempot, V.; Krysander, M.; and Staroswiecki, M. 2006. Structural analysis of fault isolability in the damadics benchmark. *"Control Engineering Practice"* 14(6):597–608.

Gertler, J. 1998. *Fault detection and diagnosis in Engineering Systems*. Marcel Dekker, Inc., Basel.

Isermann, R. 1993. Fault diagnosis of machines via parameter estimation and knowledge processing. *Automatica* 29:815–835.

Katsillis, G., and Chantler, M. 1997. Can dependency-based diagnosis cope with simultaneous equations? In *Proceedings of the Eigth International Workshop on Principles of Diagnosis (DX-97)*, 51–59.

Krysander, M. 2006. *Design and Analysis of Diagno-*

sis Systems Using Structural Methods. Ph.D. Dissertation, Linköpings universitet.

Loiez, E., and Taillibert, P. 1997. Polynomial temporal band sequences for analog diagnosis. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 474–479.

Mosterman, P.; Manders, E.; and Biswas, G. 2000. Qualitative dynamic behaviour of physical system models with algebraic loops. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX-00)*.

Mosterman, P. 1997. *Hybrid dynamic systems: a hybrid bond graph modeling paradigm and its applications in diagnosis*. Phd thesis, Vanderbilt University, Nashville, Tennessee, USA.

Patton, R. J.; Frank, P. M.; and Clark, R. N. 2000. *Issues in fault diagnosis for dynamic systems.* Springer Verlag, New York.

Puig, V.; Quevedo, J.; Escobet, T.; and Meseguer, J. 2006. Toward a better integration of passive robust interval-based fdi algorithms. In *IFAC Safeprocess'06*.

Pulido, B., and Alonso-González, C. 2004. Possible conflicts: a compilation technique for consistency-based diagnosis. *"IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics"* 34(5):2192–2206.

Pulido, B., and Alonso, C. 2000. An alternative approach to dependency-recording engines in consistency-based diagnosis. In *Artificial Intelligence: Methodology, Systems, and Applications. Ninth International Conference (AIMSA-00)*, volume 1904 of *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer Verlag. 111–120. (This is a corrected version of the work presented at DX'99).

Pulido, B., and Alonso, C. 2001. Dealing with cyclical configurations in MORDRED. In *IX Conferencia Nacional de la Asociación Española de Inteligencia Artificial, (CAEPIA-01)*, 983–992.

Pulido, B.; Alonso, C.; and Acebes, F. 2001. Lessons learned from diagnosing dynamic systems using possible conflicts and quantitative models. In *Engineering of Intelligent Systems. Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-2001)*, volume 2070 of *Lecture Notes in Artificial Intelligence*, 135–144. (Extended version of the work presented at DX-2001).

Pulido, B. 2001. *Posibles conflictos como alternativa al registro de dependencias en línea para el diagnóstico de sistemas continuos*. PhD, E.T.S.I. Informática. Universidad de Valladolid, Valladolid.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Staroswiecki, M. 2002. *Fault Diagnosis and Fault Tolerant Control*. Encyclopedia of Life Support Systems. Oxford, UK: Eolss Publishers. chapter Structural Analysis for Fault Detection and Isolation and for Fault Tolerant Control.

Struss, P. 1997. Fundamentals of model-based diagnosis of dynamic systems. In *Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence (IJCAI-97)*, 480–485.

Travé-Maussuyès, L.; Escobet, T.; and Quevedo, J. 2000. The causal qualitative fault detection and diagnosis system CAEN and its application in the gas turbine domain. In *QMFDI Vacation School*. DAMADICS Excellence Network.

Travé-Massuyès, L., and Dague, P. 2003. Common diagnostic framework specification. Deliverable report for BRIDGE Task Group MBD2, MONET2: Network of Excellence on Model Based Systems and Qualitative Reasoning, http://monet.aber.ac.uk:8080/monet/docs/tg_minutes-_and_reports/bridge/mbd_2.doc.

Washio, T.; Sakuma, M.; and Kitamura, M. 1997. A new approach to quantitative and credible diagnosis for multiple faults of components and sensors. *Artificial Intelligence* 91(1):103–130.

Finding cyclical configuration in PC calculation

```
Algorithm computeCycles(SMEM)
Begin
 for each MEM in SMEM do
  computeObservables(variables in MEM);
  for currentVariable = each variable in MEM do
    if currentVariable is not observable then
     path = {};
     detectCycles(currentVariable, currentVariable, path);
    end_if
  end_for
 end_for
End

/* Look for every possible cycle */
Algorithm detectCycles(initialVariable, currentVariable, path)
Begin
 path = path U {currentVariable};
 for i = each interpretation | currentVariable==head(i) do
  path = path U {i};
  for c = each variable | c in tail(i) AND c not OBSERVABLE
  do
   if c == initialVariable AND type(i) !="integration" then
    store path in realCycles for MEM;
    else
     detectCycles(initialVariable, c, path);
   end_if;
  end_for;
  path = path \ {i};
 end_for;
 path = \ {currentVariable};
End

/* A variable is observables if it is directly observable
or can be computed from a collection of observables in its
intepretation*/

Algorithm isObservable(variable)
Begin
 if variable belongs to OBS  then observable = true
 else
  observable = true
  for i =  each interpretation in MEM | variable == head(i) do
    for c = each variable belonging to tail(i) do
      observable = observable AND isObservable(c)
    end_for
  end_for
 end_if
 return observable
End

/* Compute the set OBSERVABLE of real observed variables or
magnitudes directly computed from observations */

Algorithm computeObservables(variables in MEM, OBSERVABLE)
Begin
 for variable = each variable in MEM do
  if (isObservable(variable)) then add variable to OBSERVABLE;
  end_if
 end_for
End
```

A Spectrum of Symbolic On-line Diagnosis Approaches *

Anika Schumann
National ICT Australia &
The Australian National University
Canberra ACT 0200, Australia
Anika.Schumann@anu.edu.au
tel: +61 2 6125 8815
fax: +61 2 6125 8651

Yannick Pencolé
LAAS-CNRS, University of Toulouse
7 avenue du Colonel Roche
31000 Toulouse, France
Yannick.Pencole@laas.fr
tel: +33 5 61 33 69 20
fax: +33 5 61 33 69 36

Sylvie Thiébaux
National ICT Australia &
The Australian National University
Canberra ACT 0200, Australia
Sylvie.Thiebaux@anu.edu.au
tel: +61 2 6125 8678
fax: +61 2 6125 8651

Abstract

This paper deals with the monitoring and diagnosis of large discrete-event systems. The problem is to determine, on-line, all faults and states that explain the flow of observations. Model-based diagnosis approaches that first compile the diagnosis information off-line suffer from space explosion, and those that operate on-line without any prior compilation have poor time performance. Our contribution is a broader spectrum of approaches that suits applications with diverse time and space requirements. Approaches on this spectrum differ in the amount of reasoning and compilation performed off-line and therefore in the way they resolve the tradeoff between the space occupied by the compiled information and the time taken to produce a diagnosis. We tackle the space and time complexity of diagnosis by encoding all approaches in a symbolic framework based on binary decision diagrams. This allows for the compact representation of the compiled diagnosis information, and for its handling across many states at once rather than for each state individually. Our experiments demonstrate the diversity and scalability of our symbolic methods spectrum, as well as its superiority over the corresponding enumerative implementations.

Introduction

There is an increasing need for automated monitoring and supervision tools for large discrete-event systems in areas as diverse as telecommunication, power distribution, manufacturing, spatial exploration, and web services. Such tools aim at assisting the operator in charge of the system supervision with tasks that include diagnosis, reconfiguration, and control.

This paper is concerned with automated diagnosis, and more specifically with the on-line identification of the faults that explain the continual flow of observations received from the system. Existing model-based approaches typically fall into two categories. In the first, a significant amount of off-line reasoning is performed to compile the system model into a larger model that embeds diagnosis information. This information, generated once and for all, is then exploited on-line to more efficiently produce the diagnosis from the actual observations. In the second category, no such compilation is performed and all the reasoning is done on-line.

The diagnoser approach (Sampath *et al.* 1996) is the archetype of compilation-based techniques. Off-line, it compiles all possible diagnoses into a finite-state machine (the diagnoser). On-line, this machine is simply run to efficiently retrieve the diagnoses explaining the current flow of observations. Unfortunately, diagnosers can be so large that they are not computable for all but the smallest applications. On-line simulation based approaches (Baroni *et al.* 1999) fall in the no-compilation camp. They directly compute the diagnosis from the behavioral model of the system by simulating possible trajectories. Here the space requirements are reasonable, but the simulation time can be excessive for large applications.

Clearly we need a more flexible resolution of the tradeoff between on-line and off-line computation, that is, between time and space. Research in that direction includes the decentralised diagnoser approach (Pencolé & Cordier 2005) which precomputes diagnosers for small subsystems only, but needs to ensure consistency of the local diagnoses at runtime. Another recent line of work deals with the incremental on-line compilation of diagnosis information and its reuse (Lamperti & Zanella 2006).

In this paper, we take an orthogonal approach to resolving the space-time tradeoff. We present a spectrum[1] of methods which differ by the degree of reasoning performed off-line and by the nature and the size of the underlying compiled models. These methods range from no compilation to full compilation of diagnosis information, but are not limited to those extreme cases.

To increase efficiency, all models are represented by symbolic finite-state machines using binary decision diagrams (BDDs), and all methods are implemented via symbolic operations. BDDs enable the compact encoding and the implicit manipulation of sets of states and transitions. On the one hand, they allow us to reduce the space requirements of models with a high degree of compilation. On the other hand, they help reducing the diagnosis time of approaches with a low degree of compilation by avoiding the individual consideration of all possible diagnosis explanations.

*This research was partly supported by National ICT Australia (NICTA) in the framework of the SuperCom project (Model-Based Supervision of Composite Systems). NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian National Research Council.

[1] This is not to be confused with the spectrum of diagnosis definitions presented in (Brusoni *et al.* 1998).

Our experiments illustrate the diversity of space-time requirements of methods across the spectrum, and clearly demonstrate the superiority of our symbolic methods over the equivalent enumerative ones.

The paper is organised as follows. After a brief reminder of BDDs and symbolic finite state machines, we present the successive models underlying the respective methods, give an on-line diagnosis algorithm for each of them, experimentally illustrate the strength of our approach, and conclude with related and future work.

Symbolic Finite State Machines

An ordered binary decision diagram (OBDD, or BDD for short) (Bryant 1986), is a rooted directed acyclic graph representation of a boolean function $\mathcal{B}^n \mapsto \mathcal{B}$. As can be seen in Figure 1, a BDD has one or two leaf nodes marked with 0 or 1, and a set of internal nodes each marked with one of the function's variables. Internal nodes have two outgoing edges: *high* and *low* edge. Given a truth assignment of the variables, the value of the function is determined by traversing the BDD top down and following the high edge if the variable marking the current node is true, and the low edge otherwise. The function's value is *true* if the leaf marked 1 is reached, and *false* otherwise.

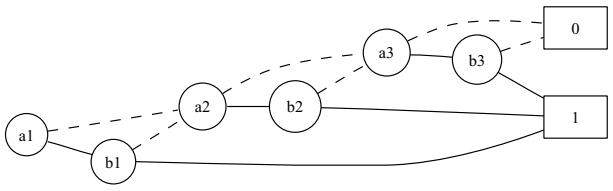


Figure 1: BDD for $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$. Dashes represent low edges.

OBDDs obey the constraint that every path in the graph respects a linear ordering of the variables. A BDD is also reduced such that (1) no two distinct nodes are marked with the same variable and have identical low and high successors, and (2) no node has identical low and high successors. These reductions make the BDD a canonical representation of the function given the chosen variable ordering. While the BDD representation still requires exponential space in the number of boolean variables in the worst case, the reductions often make the BDD of a function much smaller than its disjunctive normal form (DNF). Any boolean operation $f \star g$ on two BDDs f and g, can be carried out in $\mathbb{O}(|f||g|)$ at most, where $|f|$ denotes the number of nodes in the BDD f.

In our approach, the finite-state machines (FSMs) describing our diagnosis models are encoded symbolically, by means of BDDs, and all diagnosis algorithms are implemented in terms of BDD operations. This confers us the ability to compactly represent and efficiently manipulate sets of states and transitions.

To encode the set of states X and the set of events Σ of a FSM, it is necessary to introduce $Nr(Q) = \lceil \log_2 |Q| \rceil$ boolean variables for each set Q. Thus the events labelling the transitions can be encoded with the boolean variables $b^\Sigma = \{b_1^\Sigma, \ldots, b_{Nr(\Sigma)}^\Sigma\}$ and the states with the variables $b^X = \{b_1^X, \ldots, b_{Nr(X)}^X\}$. The initial state of the FSM is then simply given by a boolean function (represented by a BDD) over these state variables. For instance, in a 6 state FSM, the state x_2 would be given by the conjunction $\neg b_3^X \wedge b_2^X \wedge \neg b_1^X$, and the set of states $\{x_2, x_5\}$ by the DNF $(\neg b_3^X \wedge b_2^X \wedge \neg b_1^X) \vee (b_3^X \wedge \neg b_2^X \wedge b_1^X)$ which the BDD representation will reduce.

Transitions require the introduction of another set of state variables $b^{X'} = \{b_1^{X'}, \ldots, b_{Nr(X)}^{X'}\}$, called the primed variables, which are used to represent the target states of the transitions. Each transition can then be given as a conjunction involving the state variables, event variables, and primed variables. For instance, in a FSM consisting of 6 states and 3 events, the transition $t = x_2 \xrightarrow{\sigma_1} x_5$ would be given by $t = (\neg b_3^X \wedge b_2^X \wedge \neg b_1^X) \wedge (\neg b_2^\Sigma \wedge b_1^\Sigma) \wedge (b_3^{X'} \wedge \neg b_2^{X'} \wedge b_1^{X'})$. The transition relation, i.e, a set T of transitions, can then be given as a DNF which the BDD data structure will hopefully greatly reduce.

Spectrum of Symbolic Diagnosis Models

This section formally defines four symbolic models on which we base our diagnosis algorithms. These models are inspired from (Schumann, Pencolé, & Thiébaux 2004) which uses them as successive steps in the computation of a symbolic diagnoser. The models differ in the extent to which information is compiled, starting with the component models, the simple representation of the system without any precomputation, to the diagnoser model which compiles the diagnosis information for every possible sequence of observations. We choose the encodings that make use of BDDs as few as possible while still allowing an efficient on-line retrieval of diagnosis information. Efficiency requires for instance that we partition the transition sets of our FSMs.

Figure 2 shows the four models we use: the component, global, abstracted and diagnoser models and their relationship. Since the focus of the paper is the use of the models for on-line diagnosis, we will only briefly allude to their off-line computation. We refer the reader to (Schumann, Pencolé, & Thiébaux 2004) for details of how this might be done.

Component Models

As in (Sampath *et al.* 1996), the diagnosed system is composed of a set of n individual components G_i, with respective sets of states X_i, and a global event set Σ. The events are partitioned into observable Σ_o and unobservable Σ_u events, the latter of which is further partitioned into faults Σ_f and shared Σ_s events. The shared events are used to describe the communication between components.

Following the usual symbolic FSM representation described above, the symbolic components are

$$G_i = \langle b^{X_i}, b^{X_i'}, b^\Sigma, x_{0_i}, T_{o_i}, T_{s_i}, T_{f_i} \rangle,$$

where b^{X_i}, $b^{X_i'}$, and b^Σ are the Boolean variables that define the following BDDs: x_{0_i} to represent the initial state and $T_{o_i}, T_{s_i}, T_{f_i}$ to represent the observable, shared and fault

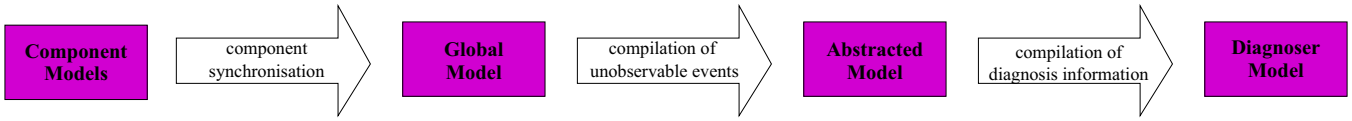


Figure 2: Relation between the diagnosis models

transitions. Note that every transition in every component G_i is defined over the same global event variables but over local state variables, that is, over $b^{X_i} \cup b^{\Sigma} \cup b^{X_i'}$.

Global Model

Diagnosing directly from the component models, without any compilation at all, is space efficient but very slow. Our second model incorporates a limited form of compilation arising from performing synchronisation off-line.

The global model is the synchronous product of the n component models: its state space is the Cartesian product of the state spaces of the components and its transitions are synchronised in that any shared event always occurs simultaneously in all components that define it. Similarly to the component models it is symbolically represented as

$$G = \langle b^X, b^{X'}, b^{\Sigma}, x_0, T_o, T_s, T_f \rangle,$$

where $b^X = \cup_{i=1}^n b^{X_i}$ (resp. $b^{X'} = \cup_{i=1}^n b^{X_i'}$) is the union of the components local state (resp. primed) variables. State $x_0 = \wedge x_{0_i}$ is the initial state. Also the BDDs T_o, T_s, T_f representing the global observable, shared and fault transitions are computed from the local transitions mainly by applying the $\wedge$ operator.

Abstracted Model

Diagnosing based on the global model is also not very efficient, since only limited information about unobservable events has been compiled away. We therefore add another model to our spectrum, the abstracted model, which is derived from the global one by abstracting all unobservable non-fault transitions and the order in which faults can occur. Hence its states $\widetilde{X}$ are obtained from the global ones, by removing all states (except the initial one) that are not the start or target state of an observable transition T_o. All sequences of unobservable events are replaced by a single transition labelled with the union of the corresponding faults (which can be empty if the sequence consists only of shared events). The set $\widetilde{T}_f$ of these new transitions is defined as

$$\Big\{ x \xrightarrow{l} x' \mid \exists \text{ path } x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_{k-1}} x_{k-1} \xrightarrow{\sigma_k} x' \text{ in } G \text{ with } x, x' \in \widetilde{X}, \sigma_1, \ldots, \sigma_k \in \Sigma_u \text{ and } l = \{\sigma_1, \ldots, \sigma_k\} \cap \Sigma_f \Big\}.$$

Figure 3 shows an example of an abstracted model. In the symbolic setting, the abstracted model

$$\widetilde{G} = \langle b^X, b^{X'}, b^{\Sigma_o}, b^F, x_0, T_o, \widetilde{T}_F \rangle$$

is encoded using the same boolean state variables as the global model, the subset b^{Σ_o} of boolean variables representing the observable events, and an additional $|\Sigma_f|$ variables $b^F = \{b_1^f, \ldots, b_{|\Sigma_f|}^f\}$ needed for the fault transition labels $F \subseteq 2^{\Sigma_f}$. There is a one to one correspondence between fault events and these variables, and a fault transition label is encoded as a conjunction of literals over b^F whose signs depend on whether the corresponding fault belongs to the label. Note that the abstracted states $\widetilde{X} \subseteq X$ are encoded over the same boolean variables as the global ones, since their number is not significantly smaller than $|X|$.

Diagnoser Model

The abstracted model still requires the on-line computation of fault information, which slows down on-line diagnosis. We therefore also consider a diagnoser model in which this entire information is compiled. A diagnoser is a deterministic finite state machine whose transitions are only labelled with observable events and whose states are directly labelled by the diagnosis information that is consistent with the past observations. This information consists of a sets of pairs (x, l) denoting a state and a fault label of the abstracted model. Let $\hat{X}$ be the set of diagnoser states, and let $\hat{x}_0$ be the initial diagnoser state. Let $\hat{R}$ denote the diagnoser state labelling function which associates a diagnoser state to the pairs in its label and verifies $\hat{R}(\hat{x}_0) = \{(x_0, \emptyset)\}$. The set $\hat{T}$ of diagnoser transitions then satisfies: $\hat{x} \xrightarrow{\sigma} \hat{x}' \in \hat{T}$ iff

$$\hat{R}(\hat{x}') = \Big\{ (x', l') \mid \exists (x, l) \in \hat{R}(\hat{x}) \text{ such that } \exists (x \xrightarrow{l''} x'') \in \widetilde{T}_F \text{ and } \exists (x'' \xrightarrow{\sigma} x') \in T_o \text{ and } l' = l \cup l'' \Big\}.$$

Figure 3 gives an example. Symbolically the diagnoser

$$\hat{G} = \langle b^{\hat{X}}, b^{\hat{X}'}, b^X, b^{\Sigma_o}, b^F, \hat{x}_0, \Phi, \hat{T} \rangle$$

is encoded using the additional variables $b^{\hat{X}}$ and $b^{\hat{X}'}$ for representing diagnoser states in their role as start and target states of transitions. The BDD Φ encodes the diagnoser state labelling function $\hat{R}$ and is defined over the variables $b^{\hat{X}} \cup b^X \cup b^F$.

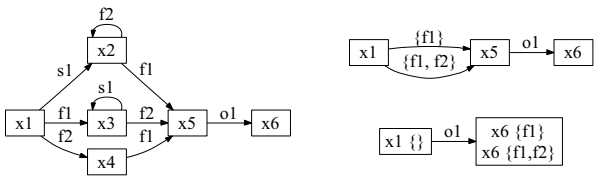


Figure 3: Global (left), abstracted (top right) and diagnoser models (bottom right). $\Sigma_o = \{o_1\}, \Sigma_s = \{s_1\}, \Sigma_f = \{f_1, f_2\}$

Symbolic On-line Diagnosis

On-line diagnosis aims to detect faults while the system is working. Given a sequence of observations, it identifies all the faults and system states that are consistent with the occurrence of these events. For each of the above models, we give a procedure that uses symbolic reasoning to compute this diagnosis information as efficiently as possible.

Initially the system is in state x_0 and no fault has occurred, so the diagnosis information is $x_0 \wedge F_\emptyset$, where $F_\emptyset = \bigwedge_{j=1}^{|\Sigma_f|} \neg b_j^f$ denotes the empty fault label. Now, each time an event σ is observed, the diagnosis information $\hat{x}'_{info}$ is derived based on σ, one of the models, and the previous diagnosis information $\hat{x}_{info}$. In this section we show how we can symbolically retrieve $\hat{x}'_{info}$ using the basic boolean operations and the following ones:

- IsDef(bdd) returns true iff bdd does not represent *false*
- Extract(bdd, B)
  deletes from bdd all occurrences of variables not in B,
- Abstract(bdd, B)
  deletes from bdd all occurrences of variables in B,
- Swap($bdd,\ \{a_1, \ldots, a_k\}, \{b_1, \ldots, b_k\}$) renames, in bdd, variable a_i with b_i, $i = 1 \ldots k$, and vice versa.

For the sake of readability, the algorithms are presented in the following order from the diagnoser to the component based one.

On-line diagnosis based on the diagnoser

The precomputed diagnoser contains all the information to perform efficient on-line diagnosis. Given the previous diagnoser state $\hat{x}$ and a new observation σ it is sufficient to

1. trigger the corresponding transition and
2. retrieve the fault information from its target state.

Algorithm 1 describes the symbolic procedure. To trigger the transition (step 1), we apply three BDD operations, namely $\wedge$, $Extract$, and $Swap$. Applying the $\wedge$ operation to the encodings of a start state $\hat{x}$, an event σ and the transition set $\hat{T}$, retrieves the transition that starts in $\hat{x}$ and is labelled with σ. Next, the operation $Extract$ is used to obtain only the target state $\hat{x}'$ of the transition. We then $Swap$ the encoding of $\hat{x}'$ over the primed variables for an encoding over the non-primed ones in order to determine its label and to trigger future transitions. To determine the label of $\hat{x}'$ (step 2), we first conjoin $\hat{x}'$ with the state labelling function Φ, and then abstract from the boolean variables representing diagnoser states.

Algorithm 1 $DiagDiagnose(\hat{G}, \hat{x}, \sigma)$

1: $\hat{x} \leftarrow \hat{x} \wedge \sigma \wedge \hat{T}$
   $\hat{x}' \leftarrow \text{Extract}(\hat{x}, b^{\hat{X}'})$
   $\hat{x}' \leftarrow \text{Swap}(\hat{x}', b^{\hat{X}}, b^{\hat{X}'})$
2: $\hat{x}'_{info} \leftarrow \hat{x}' \wedge \Phi$
   return Abstract $(\hat{x}'_{info}, b^{\hat{X}})$

On-line diagnosis based on the abstracted model

Using the abstracted model, the retrieval of the diagnosis information given its predecessor $\hat{x}_{info}$ requires:

1. computing the states $\widetilde{X}_{unObs}$ that can be reached from those contained in $\hat{x}_{info}$ before observing the new event σ,
2. computing the fault labels representing the faults that have occurred on a path from the initial state to a state in $\widetilde{X}_{unObs}$, and
3. triggering all transitions starting from states in $\widetilde{X}_{unObs}$ and labelled σ.

The corresponding three symbolic computation steps are shown in Algorithm 2. Once the unobservable transitions $\widetilde{T}_{unObs}$ starting in a state of $\hat{x}_{info}$ are determined (line 1), they contain all the new faults that could have occurred since the last observation. These are added to the faults in $\hat{x}_{info}$ that have previously occurred using function $AddFault$. Symbolically, adding a fault f_i implies changing the value of the corresponding boolean variable b_i^f from false to true. It is done by abstracting b_i^f from the fault label l (i.e. $Abstract(l, \{b_i^f\})$) and conjoining it with l (i.e. $l \wedge b_i^f$). This abstraction can be done simultaneously for all fault labels L_{f_i} to which f_i has to be added.

Finally the observable transitions are triggered and the new diagnosis information returned (step 3).

Algorithm 2 $AbstDiagnose(\widetilde{G}, \hat{x}_{info}, \sigma)$

1: $\widetilde{T}_{unObs} \leftarrow \widetilde{T}_f \wedge Extract(\hat{x}_{info}, b^X)$
2: $\widetilde{X}_{unObs} \leftarrow \hat{x}_{info} \vee AddFault(\widetilde{T}_{unObs}, \hat{x}_{info})$
   $\widetilde{X}_{unObs} \leftarrow Swap(\widetilde{X}_{unObs}, b^X, b^{X'})$
3: $\hat{x}'_{info} \leftarrow Extract(\widetilde{X}_{unObs} \wedge \sigma \wedge T_o, b^{X'} \cup b^F)$
   return $Swap(\hat{x}'_{info}, b^X, b^{X'})$

On-line diagnosis based on the global model

Using the global model, the symbolic computation of the diagnosis information is similar to that above, except that states $\widetilde{X}_{unObs}$ now need to be computed based on transition sequences in G. For this purpose, we first combine shared and fault transitions into a single transition set T_u, in which all events are defined over variables b^F. Here shared transitions are labelled with the empty fault label $F_\emptyset$.

Algorithm 3 describes the symbolic procedure. All formulas $\widetilde{X}_{unObs}$, X_{new}, and X_{targ} represent sets of labelled states, that is, sets of tuples (x, l). $\widetilde{X}_{unObs}$ is computed using breadth-first search (lines 1-8). Initially $\widetilde{X}_{unObs}$ and X_{new} are composed of the previous diagnosis information $\hat{x}_{info}$ (lines 1-2). As long as there are still new diagnosis tuples X_{new} that have not been processed (line 3), applicable unobservable transitions are triggered (line 4) and any fault labelling them is added (line 5). The tuples already closed are removed from the resulting tuples X_{targ} to ensure the

termination of the algorithm (operator $\wedge\neg$ in line 6). The new tuples are added to the set of closed ones (operator $\vee$ in line 7). Once $\widetilde{X}_{unObs}$ is obtained, the new diagnosis information is retrieved as in step 3 of Algorithm 2 (line 9).

Algorithm 3 $GlobDiagnose(T_u, T_o, \hat{x}_{info}, \sigma)$

1: $X_{new} \leftarrow \hat{x}_{info}$
2: $\widetilde{X}_{unObs} \leftarrow X_{new}$
3: **while** $IsDef(X_{new})$ **do**
4: $\quad T_{new} \leftarrow T_u \wedge Extract(X_{new}, b^X)$
5: $\quad X_{targ} \leftarrow AddFault(T_{new}, X_{new})$
$\quad X_{targ} \leftarrow Swap(X_{targ}, b^X, b^{X'})$
6: $\quad X_{new} \leftarrow X_{targ} \wedge \neg\widetilde{X}_{unObs}$
7: $\quad \widetilde{X}_{unObs} \leftarrow \widetilde{X}_{unObs} \vee X_{new}$
8: **end while**
9: $\hat{x}'_{info} \leftarrow Extract(\widetilde{X}_{unObs} \wedge \sigma \wedge T_o, b^{X'} \cup b^F)$
return $Swap(\hat{x}'_{info}, b^X, b^{X'})$

On-line diagnosis based on the component models

In addition to the previous algorithm, on-line diagnosis based on the component models requires the computation of those of the global transitions that are needed to determine the new diagnosis information. For every component G_i we only need to consider

- all sequences of unobservable transitions T_{u_i} starting in a state $x_i \in X'_i$ consistent with the previous diagnosis information ($X'_i = Extract(\hat{x}_{info}, b^X_i)$) and
- all observable transitions T_{σ_i} labelled with the new observation σ ($T_{\sigma_i} = \sigma \wedge T_{o_i}$).

To obtain the corresponding global transitions efficiently, via the $\wedge$ operator, a synchronous product is required. In a synchronous system, when a transition is triggered in a component G_i, a transition is also triggered in every other component. Hence we add for every event σ' that can occur in G but not in G_i and every state x of a component model G_i, a transition $x \xrightarrow{\sigma'} x$. Now the relevant global transitions are computed as follows:

- $T_u \leftarrow \wedge_{i=1}^n T_{u_i}$ and similarly
- $T_\sigma \leftarrow \wedge_{i=1}^n T_{\sigma_i}$.

Using these two transition sets, the new diagnosis information is computed as in Algorithm 3. The only change needed is the replacement of $\sigma \wedge T_o$ with T_σ in line 9 of the algorithm.

Experimental Evaluation

Our approach has been implemented on top of the CUDD BDD package.[2] In order to evaluate the benefits of our symbolic framework, we also implemented a traditional "enumerative" version of the models and algorithms, using optimised automata data structures which facilitate the manipulation of individual states and transitions. These two implementations enable us to present experimental evidence

[2] `http://vlsi.colorado.edu/~fabio/CUDD`

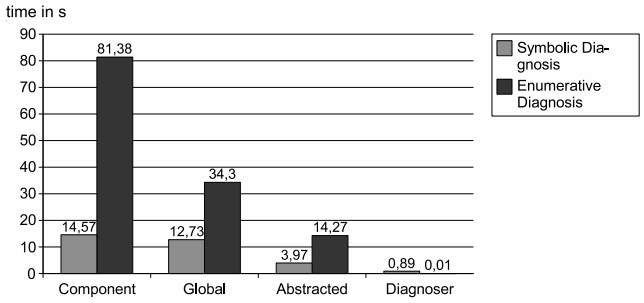


Figure 4: Average diagnosis times over 100 scenarios of 10000 observations each.

that the symbolic approach yields important gains in time or space.

Our experiments below were run on a 1.2 GHz Pentium IV with 512 Mb of memory. We first use the largest example in (Schumann, Pencolé, & Thiébaux 2004), which is derived from a telecommunication application. It consists of 3 components (a switch with 12 states and 18 transitions, a primary control station of 13 states and 15 transitions, and a backup control station of 19 states and 28 transitions), 9 observable events, 11 fault types, and 8 other unobservable events. We generated by simulation 100 arbitrary scenarios (possible sequences of observations) of 10000 observations each, and used them as input to all models.

Figure 4 compares the time performance of the various on-line diagnosis methods. All symbolic models except the diagnoser are more efficient to use than their enumerative counterparts. This should not come as a surprise: BDDs are well suited to triggering transition sets and enable the consideration of all diagnosis tuples at once, but do not generally pay off when only a single transition is involved as is the case with the diagnoser.

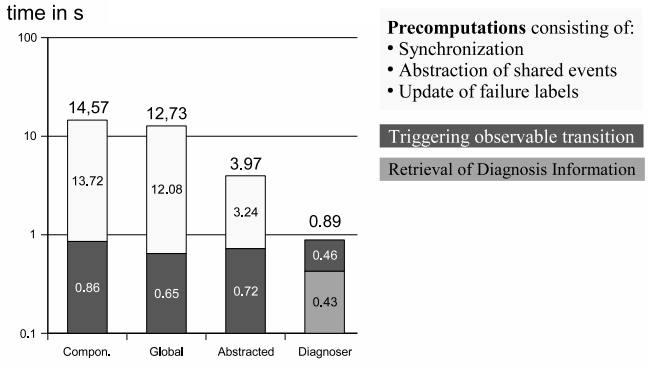


Figure 5: Composition of the diagnosis times

Figure 5 shows the composition of diagnosis times across the symbolic spectrum. We can state that the "precomputations", that is the computation of the states that can be reached before observing the next event (previously refered to as as $\widetilde{X}_{unObs}$) account for most of the computation time. The differences in diagnosis times correlate with the extent

to which the accumulation of faults (function $AddFault$ described on page 4) is performed on-line. Even though a fault f_i can be simultaneously added to all fault labels L_{f_i}, $AddFault$ still requires the individual consideration of fault labels (in general, $L_{f_i} \neq L_{f_j}$ for $f_i \neq f_j$), which is the main bottleneck of the symbolic computation. The component and global models yield similar diagnosis times because $AddFault$ is applied the same number of times in both cases and symbolic synchronization is very fast. In contrast, the abstracted model yields significantly faster diagnosis times because $AddFault$ only needs to be applied once per observation. With the diagnoser, $AddFault$ is never called.

Taken in conjunction with the diagnosis times, the corresponding model sizes (see Table 1) illustrate the time/space tradeoff of the methods across the spectrum and the superiority of the symbolic approach. Comparing the symbolic models (resp. the enumerative ones), we can state, that the faster the on-line diagnosis based on a model, the larger the model size. For all models, the symbolic representation is as small as or smaller than the enumerative one; yet except for the diagnoser, the symbolic run-times are significantly better. Importantly, the symbolic diagnoser is 20 times smaller than the enumerative one. Its size is rather comparable to that of the enumerative abstracted model, yet it is an order of magnitude faster than the latter.

	G_i	G	$\widetilde{G}$	$\hat{G}$
states Nr.	∅ 17.7	1063	965	18474
transition Nr.	∅ 34	2912	48958	120698
space symb. (Mb)	0.01	0.2	0.6	7.5
space enum. (Mb)	0.01	0.2	2.7	123.9

Table 1: Model sizes

Focusing on the symbolic spectrum, the abstracted model appears to provide a particularly interesting tradeoff. It is 13 times smaller than the diagnoser but only 4 times slower. Compared to the global model, the percentage decrease of diagnosis time of the abstracted model is slightly higher than its percentage increase in size. The advantage of the abstracted model results from the efficiency of (1) the symbolic triggering of sets of transitions, and (2) the update of fault labels by considering fault sets rather than by considering a sequence of individual faults.

The component model also presents an interesting tradeoff due to its very small size of only 8 kilobytes. For large applications, it appears to be the only option. We show how the component-based approach scales as the size of the system increases, using a grid of computer nodes inspired from the example in (Rintanen & Grastien 2007). All nodes have the same behaviour. In normal mode, each node performs its task, sending an *on* message to a supervisor prior to starting and an *off* message upon completion. When a node becomes faulty, an automatic recovery system forces the node to reboot and to send his neighbours reboot requests which get propagated through the grid.

The model of a node has 14 states, 67 transitions, 1 fault, 8 shared, and 2 observable events. The global model of a grid of size $n \times m$ closely approaches the $14^{m \times n}$ states bound. E.g., the 2×2 grid has $14^{3.85} \simeq 26,000$ states. The example is poorly diagnosable. Every system state can be associated with the 2^{m*n} fault hypotheses, and the observations do not allow discrimination between faults due to a masking phenomenon (nodes reboot silently and reboot requests from other nodes are not observed). Consequently, there is a huge set of diagnoses that explains a given observation sequence.

Figure 6 compares the performance of the symbolic and enumerative approaches as the size of the grid increases (note the logarithmic scale). The gap between the two approaches increases by an order of magnitude with each addition of a new component. For the three larger grids, the enumerative approach failed to refine the diagnosis within an average of 10 sec – the theoretical number of diagnoses for the 2×2 grid is 2 458 624. All other enumerative approaches are unsuitable, as the enumerative global model could not be computed. In contrast, the symbolic approach was able to refine the same diagnosis in 0.079 sec. With the largest grid, which has 481 890 304 possible diagnoses, the enumerative approach could not terminate after a day of computation. The symbolic approach was able to refine the diagnosis within an average of 7.11 sec.

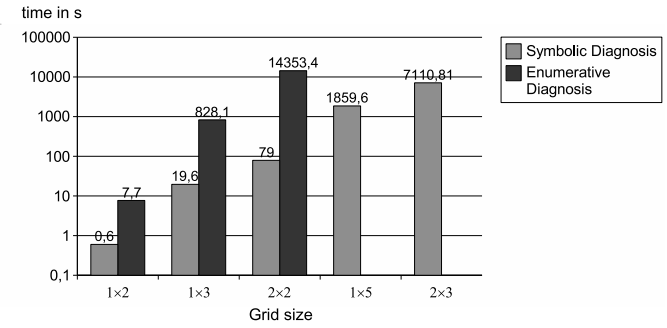


Figure 6: Average component-based diagnosis times over 10 scenarios of 1000 observations each

Related Work

There are only few other works presenting results obtained with generic on-line diagnosis software. The software presented in (Pencolé 2000) is a demonstrator of a Sampath's diagnoser which works only for small models. The UMDES Library[3] provides an enumerative implementation of Sampath's diagnoser (Sampath *et al.* 1996). UMDES cannot compete with either our symbolic or enumerative implementations. In fact, one of our motivations to implement our own enumerative algorithms was that UMDES was unable to compute the diagnoser for the smallest of the examples given in (Schumann, Pencolé, & Thiébaux 2004).

There are other softwares for diagnosing discrete-event systems. D-Dyp (Pencolé, Cordier, & Rozé 2002) is a software that implements a tool for diagnosing event trajectories using an enumerative and decentralised approach. EDEN

[3] `http://www.eecs.umich.edu/umdes/`

(Lamperti & Zanella 2003) is a software environment for diagnosing discrete-event systems off-line where the tradeoff between space and time complexity is not an essential issue.

The idea of exploiting symbolic representations in the context of discrete-event systems diagnosis is not new but it has traditionally been applied to different problems, e.g. checking diagnosability (Cimatti, Pecheur, & Cavada 2003; Rintanen & Grastien 2007), off-line diagnosis using off-the-shelf model-checkers (Cordier & Largouët 2001), or computing a symbolic diagnoser (Marchand & Rozé 2002; Schumann, Pencolé, & Thiébaux 2004). In contrast, we exploit the power of the symbolic representation to design a range of efficient on-line diagnosis approaches.

Symbolic representations based on Decomposition Negation Normal Forms (DNNFs) have successfully been applied to diagnosing *static* systems (see e.g., (Darwiche 1998)). For diagnosing *dynamic* systems however, BDDs are better suited because the main operation, namely the triggering of transitions, can be performed in polynomial time in the size of the BDD while it would require exponential time in the size of the DNNF.

In (Pencolé & Cordier 2005) the authors resolve the time/space complexity tradeoff using a single approach which merges, on-line, the results of a set of diagnosers compiled for small subsystems. This approach is orthogonal to ours. To solve the time/space complexity in such an approach, the model of the system is decentralised in order to minimize the interactions between the local diagnosers at monitoring time.

In (Lamperti & Zanella 2006), another spectrum of diagnosis algorithms is introduced. This spectrum of machines result from the compilation of knowledge that is performed when the diagnosis algorithm is running on several diagnostic problems. The idea is to compile a model at diagnosis time that contains the diagnosis for a given set of observations in order to reuse this compiled model in a similar case in the future.

Conclusion & Future Work

We have presented a spectrum of symbolic diagnosis approaches which differ in the amount of model compilation performed off-line. The underlying models range from the small component models that do not incorporate any compilation, to the diagnoser model in which the diagnosis information is compiled for the entire observable behaviour of the system. The abstracted model constitutes an interesting alternative to the diagnoser: it is considerably smaller but not much slower and so applies to a wider range of applications.

Thanks to the symbolic implementation, we are able to handle large sets of transitions and diagnosis hypotheses at once. This leads to a simple and efficient way of obtaining the correct and complete set of diagnoses. In comparison to an enumerative implementation, only the on-line use of the symbolic diagnoser incurs a small time overhead. In all other cases the run-time of the symbolic approach is significantly reduced, and so are the space requirements of the larger models. Therefore, an enumerative approach is mainly useful for very small applications for which the computation and storage of the large diagnoser is feasible.

We first plan to extend our symbolic spectrum to decentralised approaches, such as the one of (Pencolé & Cordier 2005). We actually believe that, since our monitoring algorithms are more efficient than the enumerative ones, we can use them to implement local diagnosers on bigger subsystems so that the number of on-line merging operations is reduced. Moreover, the merging operator used in the decentralised approach can be more efficienly implemented with the symbolic synchronisation operator that is used in our spectrum of algorithms. Another advantage to mix our spectrum with a decentralised approach is that the set of local diagnosers may be implemented with different algorithms of the spectrum. In fact, depending on the way the system is decentralised, the subsystems may have different requirements due to their size, their diagnosability, thus, the optimal algorithm for monitoring and diagnosing one subsystem may not be the optimal one for another subsystem.

A second perspective to this work is to use the symbolic spectrum to compute and use a set of accurate diagnosers (Pencolé, Kamenetsky, & Schumann 2006). An accurate diagnoser is a diagnosis algorithm that focuses on the diagnosis of one type of fault only with the guarantee that the diagnosis of this fault is consistent with the one provided by a global diagnoser. The accurracy of such a diagnoser relies on a specific subsystem which has its own requirements for diagnosis. With help of our spectrum, it will be possible to select the optimal diagnosis algorithm for implementing every accurate diagnoser and for increasing the scalability of this method.

Another line of future work is to extend our framework to stochastic systems and compute probability distributions on diagnoses, using for instance algebraic decision diagrams which are generalisation of BDDs to real-valued functions over the booleans. Finally, integrating diagnosis and planning for repair or reconfiguration actions is one of the most significant challenges faced by the field of model-based diagnosis (Console & Dressler 1999). Given the recent success of planning techniques based on symbolic model-checking, we believe that our framework will prove a good basis for addressing this challenge.

References

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

Brusoni, V.; Console, L.; Terenziani, P.; and Dupre, D. T. 1998. A spectrum of definitions for temporal model-based diagnosis. *Artificial Intelligence* 102(1):39–79.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. In *Proc. IJCAI-03*, 363–369.

Console, L., and Dressler, O. 1999. Model-based diagnosis in the real world: lessons learned and challenges remaining. In *Proc. IJCAI-99*.

Cordier, M.-O., and Largouët, C. 2001. Using model-checking techniques for diagnosing discrete-event systems. In *Proc. DX-01*, 39–46.

Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

Lamperti, G., and Zanella, M. 2003. Eden: An intelligent software environment for diagnosis of discrete-event systems. *Applied Intelligence* 18:55–77.

Lamperti, G., and Zanella, M. 2006. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence* 170:232–297.

Marchand, H., and Rozé, L. 2002. Diagnostic de pannes sur des systèmes à événements discrets: une approche à base de modèles symboliques. In *13ème Congrès AFRIF-AFIA de Reconnaissances des Formes et Intelligence Artificielle*, 191–200.

Pencolé, Y., and Cordier, M. O. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence* 164:121–170.

Pencolé, Y.; Cordier, M.-O.; and Rozé, L. 2002. A decentralized model-based diagnostic tool for complex systems. *International Journal on Artificial Intelligence Tools (IJAIT)* 1(3):327–346.

Pencolé, Y.; Kamenetsky, D.; and Schumann, A. 2006. Towards low-cost diagnosis of component-based systems. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Process (SAFEPROCESS)*.

Pencolé, Y. 2000. Dyp: a centralized diagnoser approach demonstration software. In *Newsletter (CDROM)*. MONET, European Network of Excellence in Model-Based Systems and Qualitative Reasoning.

Rintanen, J., and Grastien, A. 2007. Diagnosability testing with satisfiability algorithms. In *Proc. IJCAI-07*, 532–537.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems Technology* 4(2):105–124.

Schumann, A.; Pencolé, Y.; and Thiébaux, S. 2004. Diagnosis of discrete-event systems using binary decision diagrams. In *Proc. DX-04*, 197–202.

Computation of Minimal Sensor Sets from Precompiled Discriminability Relations

Gianluca Torta and **Pietro Torasso**
Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino (Italy)

Abstract

In the present paper we address the problem of computing the Minimal Sensor Sets that guarantee a desired level of diagnostic discrimination for a system.
Unlike other approaches previously proposed in the literature, we compute MSSs starting from precompiled discriminability relations that are parsimoniously stored using a symbolic representation. Such discriminability relations are built from an extended model of the system to be diagnosed which includes a set of switches modeling the inclusion (or the exclusion) of potentially observable variables into the set of actual observations.
The main advantage of the approach is that the precompiled relations can be reused for efficiently computing MSSs under different discriminability requirements and constraints on the selection of measurement points. This possibility can be exploited for conveniently analyzing different scenarios at design time, as well as for reacting to changes in requirements and constraints that happen after the MSSs have been computed.
Encouraging experimental results collected from the application of the proposed solution to the model of a non-trivial combinatorial digital circuit are reported.

Introduction

The problem of computing a set of sensors that guarantees diagnosability of a given system is well known in the Model-Based Diagnosis community.
To add to the complexity of the problem, we are often interested in finding a sensor set (is one exists) that is *minimal* according to some criteria such as set inclusion, cardinality or total cost. Moreover, it is often the case that perfect diagnosability is not achievable or even desired; in such cases, the goal is to find the minimal sensor set that satisfies the *desired* level of diagnosability.

The search space for the minimal sensor sets is usually specified by the system modeler as a set of potential measurement points, i.e. physical quantities that could be measured by placing a suitable sensor. Such a search space could be constrained, besides by a-priori physical constraints, either by positive information (e.g. we already have some sensors in place that "come for free") or negative information (e.g. we have discovered that sensors placed in certain places are too unreliable or prone to failure).

In this paper we propose a novel method for computing minimal sensor sets given a *desired* level of diagnosability and positive/negative observability constraints.
Instead of starting the computation from scratch for each given set of diagnosability requirements and observability constraints, we propose to precompile reusable discriminability relations that can be efficiently computed and parsimoniously stored using Ordered Binary Decision Diagrams (OBDDs) as a symbolic representation.
The usefulness of the precompiled discriminability relations comes from the fact that they are parametric in the system observability, i.e. for each discriminable pair of faults they specify all the possible degrees of system observability that guarantee discrimination of the two faults.

The main advantage of computing minimal sensor sets from the precompiled relations is that it makes more convenient to analyze different scenarios at design time, as well as to react to changes in requirements and constraints that happen after the MSSs have been computed.

The paper is structured as follows. First, we give the definitions of discriminability, minimal sensor sets and discriminability relations parametric in terms of system observability. We then provide an overview of the global computational process for determining the Minimal Sensor Sets, describing in detail the actual computation of the discriminability relations and of the MSSs.
Next, we briefly discuss how the computation of MSSs can be efficiently performed by exploiting OBDDs and we present encouraging experimental results collected from the application of the proposed solution to the model of a non-trivial combinatorial digital circuit. Finally, we discuss related work and summarize the contributions of the paper.

Indiscriminability and Minimal Sensor Sets

In this section we provide the formal setting for characterizing the notion of diagnostic discriminability and the one of Minimal Sensor Sets. We start from the definition of *System Description*.

Definition 1 *A* System Description *is a pair SD = (*$\mathcal{SV}$*, DT) where:*

- $\mathcal{SV}$ *is the set of discrete system variables partitioned in* P *(system ports),* C *(system components) and* I *(internal variables). The set of system ports is partitioned into* P_{exo}

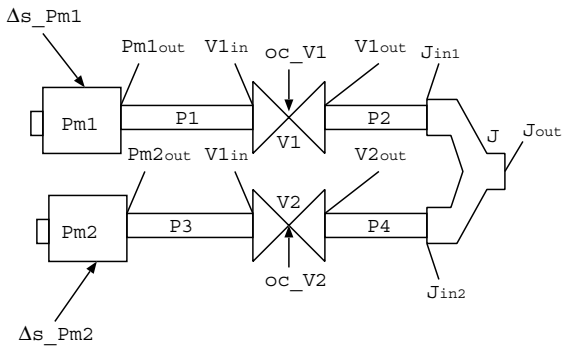


Figure 1: The Sample Hydraulic System.

$Pipe(ok)$	$\Delta f_{out} = \Delta f_{in}\ ;\ \Delta r_{in} = \Delta r_{out}$
$Pipe(lk)$	$\Delta f_{out} = \Delta f_{in} \oplus -\ ;\ \Delta r_{in} = \Delta r_{out} \oplus -$
$Pipe(pc)$	$\Delta f_{out} = \Delta f_{in}\ ;\ \Delta r_{in} = \Delta r_{out} \oplus +$
$Pipe(cl)$	$\Delta f_{out} = \Delta f_{in} = -\ ;\ \Delta r_{in} = +$
$Pump(ok)$	$\Delta f_{out} = \Delta s \ominus \Delta r$
$Pump(up)$	$\Delta f_{out} = \Delta s \ominus \Delta r \oplus -$
$Pump(op)$	$\Delta f_{out} = \Delta s \ominus \Delta r \oplus +$
$Join(ok)$	$\Delta f_{out} = \Delta f_{in1} \oplus \Delta f_{in2}\ ;\ \Delta r_{out} = \Delta r_{in1} = \Delta r_{in2}$
$Valve(ok)$	$oc(open) \Rightarrow \Delta f_{out} = \Delta f_{in}\ ;\ \Delta r_{in} = \Delta r_{out}$
	$oc(close) \Rightarrow \Delta f_{out} = \Delta f_{in} = -\ ;\ \Delta r_{in} = +$
$Valve(so)$	$\Delta f_{out} = \Delta f_{in}\ ;\ \Delta r_{in} = \Delta r_{out}$
$Valve(sc)$	$\Delta f_{out} = \Delta f_{in} = -\ ;\ \Delta r_{in} = +$

Figure 2: Qualitative Equations of the Behavior of Hydraulic Components.

(exogenous ports) and P_{end} (endogenous ports).
We will denote with $dom(v)$ the finite domain of variable $v \in \mathcal{SV}$; in particular, for each $c \in C$, $dom(c)$ consists of the list of possible behavioral modes *for c (an* ok *mode and one or more fault modes)*

- DT (Domain Theory) is a relation over the variables in $\mathcal{SV}$[1]

Example 1 *As a running example we will consider the simple hydraulic system reported in Figure 1.*

The models of a generic pump, valve, join and pipe are reported in Figure 2. The models are expressed in terms of qualitative equations involving qualitative deviations (Struss, Malik, & Sachenbacher 1996): for example, when a pipe is in the behavioral mode pc (partially clogged), the qualitative deviation Δf_{out} of the flow at the end of the pipe is the same as the qualitative deviation Δf_{in} of the flow at the beginning of the pipe. For the pressure, instead, we have that $\Delta r_{in} = \Delta r_{out} \oplus +$, i.e. the deviation of the pressure is qualitatively increased at one end of the pipe with respect to the other end.
Note that it is straightforward to translate the domain model of a component expressed via qualitative equations into relational form by taking into account the semantics of sign algebra.

The hydraulic system has four exogenous ports: the expected operating conditions oc_V1 and oc_V2 of the two valves (each valve could be open or closed) and the qualitative deviations Δs_Pm1 and Δs_Pm2 of the control signals of the two pumps (that influence the quantity of fluid pumped by the pumps). Therefore, $P_{exo} = \{\Delta s\_Pm1,\ \Delta s\_Pm2,\ oc\_V1,\ oc\_V2\}$.

As concerns the endogenous ports, we have modeled the qualitative deviations of the flow and pressure at each of the connection points that have an associated label in Figure 2; for example, for connection point $Pm1_{out}$, we have modeled variables Δf_Pm1_{out} and Δr_Pm1_{out}. In this way P_{end} contains 18 potentially observable variables.

The notion of discriminability we will consider in this paper is based on assumption that the the system to be diagnosed can be tested under different contextual conditions in order to find the set of consistency-based diagnoses (see e.g. (Esser & Struss 2007)).

We are now ready for introducing the notion of discriminability and MSS. In particular we start from the characterization of discriminability among instances of pairs of components.

Definition 2 *Let $c_1, c_2 \in C$ and $bm_i, bm_k \in dom(c_1)$, $bm_j, bm_l \in dom(c_2)$. We say that $(c_1(bm_i), c_2(bm_j))$ and $(c_1(bm_k), c_2(bm_l))$ are discriminable w.r.t. observability $O \subseteq P_{end}$ iff there exists an instance $\mathcal{X}$ of P_{exo} s.t. for each instance $\mathcal{C}_{\overline{1,2}}$ of $C\backslash\{c1, c2\}$:*

$$\Pi_O(\sigma_{c_1(bm_i),c_2(bm_j)}(DT \bowtie \mathcal{C}_{\overline{1,2}} \bowtie \mathcal{X})) \cap$$
$$\cap \Pi_O(\sigma_{c_1(bm_k),c_2(bm_l)}(DT \bowtie \mathcal{C}_{\overline{1,2}} \bowtie \mathcal{X}) = \emptyset$$

where Π, σ and $\bowtie$ are the project, select *and* join *operations defined in the relational algebra.*

According to the above definition $(c_1(bm_i), c_2(bm_j))$ and $(c_1(bm_k), c_2(bm_l))$ are considered discriminable w.r.t. a given observability O when we can find an instance $\mathcal{X}$ of the inputs such that the two instances of c_1, c_2 *necessarily* induce different values for the observable variables O regardless of the status $\mathcal{C}_{\overline{1,2}}$ of the other components.

It is worth pointing out that the requirement of necessarily obtaining different values for the observables is a direct consequence of the choice of modelling DT as a relational model. In fact, the absence of a functional dependency of the observables on the behavioral modes of the components and the exogenous ports requires that the observations in $\Pi_O(\sigma_{c_1(bm_i),c_2(bm_j)}(DT \bowtie \mathcal{C}_{\overline{1,2}} \bowtie \mathcal{X}))$ are disjoint from the ones in $\Pi_O(\sigma_{c_1(bm_k),c_2(bm_l)}(DT \bowtie \mathcal{C}_{\overline{1,2}} \bowtie \mathcal{X})$.

Example 2 *The maximum observability of our example system involves all of the 18 variables in P_{end}. Unfortunately, even if we could observe all the endogenous ports, the direct application of the notion of discriminability of Definition 3 yields no discriminable pairs of component instances; this is due to the masking effect exhibited by most of the possible faults. For the sake of our illustrative purposes, we therefore assume that at most one component can be faulty.*
Let us assume that the observability O in our example system is $\{\Delta f\_Pm1_{out},\ \Delta r\_Pm1_{out},\ \Delta f\_V1_{in},\ \Delta r\_V1_{in},\ \Delta f\_V1_{out},\ \Delta r\_V1_{out}\}$, i.e. we observe the qualitative de-

[1] Usually, in component-based systems, relation DT is obtained by joining a set of relations $DT_1, \ldots, DT_n$ each one modeling the behavior of a component.

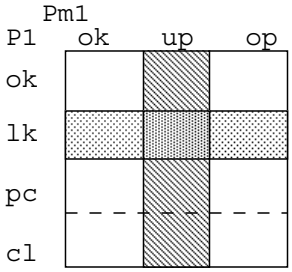


Figure 3: Discriminability of $Pm1(up)$ and $P1(lk)$.

viations of the flow and pressure at the output of $Pm1$ *and at the two ends of* $V1$.
It turns out that the assignments $(Pm1(up), P1(ok))$ *and* $(Pm1(ok), P1(lk))$ *are discriminable w.r.t. O, i.e. there exists an assignment to the exogenous ports of the system s.t. the possible observations induced by* $(Pm1(up), P1(ok))$ *(with all other components being* ok *given to our single-fault assumption) are disjoint from the possible observations induced by* $(Pm1(ok), P1(lk))$.
In particular, the discrimination is possible when we set the exogenous ports of the system so that $V1$ *is open (i.e.* $oc\_V1(open)$*) and the control signal of* $Pm1$ *is not below nominal (i.e.* $\Delta s\_Pm1(0)$ *or* $\Delta s\_Pm1(+)$*).*

Definition 3 *Let* $c_1, c_2 \in C$ *and* $bm \in dom(c_1)$, $bm' \in dom(c_2)$. *We say that* $c_1(bm)$ *and* $c_2(bm')$ *are discriminable w.r.t.* $O \subseteq P_{end}$ *iff* $(c_1(bm_i), c_2(bm_j))$ *and* $(c_1(bm_k), c_2(bm_l))$ *are discriminable w.r.t. O where* bm_i, bm_j bm_k bm_l *obey one of the following constraints:*

1. $bm_i = bm \wedge bm_j = bm' \wedge (bm_k \neq bm \vee bm_l \neq bm')$
2. $bm_i = bm \wedge bm_j \neq bm' \wedge bm_k \neq bm$
3. $bm_i \neq bm \wedge bm_j = bm' \wedge bm_l \neq bm'$

Example 3 *The three cases of Definition 3 can be better illustrated by looking at Figure 3, where behavioral modes* up *of* $Pm1$ *and* lk *of* $P1$ *identify a cross in the matrix* $dom(Pm1) \times dom(P1)$.
Case 1 *ensures that we discriminate the presence of* $(Pm1(up), P1(lk))$ *by requiring that the center of the cross is compared to all of the other cells; case* 2 *ensures that we discriminate the presence of* $Pm1(up)$ *by requiring that each cell on the vertical arm of the cross is compared to all of the cells outside such arm; discrimination of* $P1(lk)$ *is handled in a similar way by case* 3.
It turns out that, under the observability and single-fault assumption of example 2, the assignments $Pm1(up)$ *and* $P1(lk)$ *are discriminable, i.e. our diagnostic system is able to distinguish between the situations where* $Pm1$ *is underpumping and those where* $P1$ *is leaking.*
For reaching such a conclusion, it is necessary to check the joint discriminability of many pairs of instances of $Pm1$ *and* $P1$ *according to the cases of Definition 3: considering case 2,* $(Pm1(up), P1(ok))$ *must be discriminable from* $(Pm1(ok), P1(ok))$, $(Pm1(ok), P1(lk))$, …, $(Pm1(op), P1(ok))$, …*; considering case 3,* $(Pm1(ok), P1(lk))$ *must be discriminable from* $(Pm1(ok), P1(ok))$, $(Pm1(up), P1(ok))$, …, $(Pm1(ok), P1(pc))$, …. *Case 1, in which* $Pm1(up)$ *and* $P1(lk)$*, is forbidden by the single-fault assumption made for our example.*

In general we are interested in verifying whether a given set of fault discriminations are satisfied and what kind of observability guarantees such a discrimination. For this reason we introduce the following two notions.

Definition 4 *A discriminability requirement* δ *is a pair* $(c_1(bm), c_2(bm'))$. *We denote as* $\Delta = \{\delta_1, \ldots, \delta_m\}$ *a set of discriminability requirements.*

Definition 5 *Given a System Description SD = (*$\mathcal{SV}$*, DT), an observability O and a set of discriminability requirements* $\Delta = \{\delta_1, \ldots, \delta_m\}$*, we say that O* satisfies Δ *if for each* $\delta_i = (c_1(bm), c_2(bm')) \in \Delta$, $c_1(bm)$ *and* $c_2(bm')$ *are discriminable w.r.t. O.*
A Minimal Sensor Set *(w.r.t.* Δ*) is an observability* O^* *such that no other observability* O', $|O'| < |O^*|$ *satisfies* Δ.

From the definition above it is apparent that the preference criterion chosen for selecting Minimal Sensor Sets is based on the minimum cardinality. This reflects the assumption that all the sensors have an equal cost.

Example 4 *It turns out that the observability O assumed in the previous examples (namely* $\{\Delta f\_Pm1_{out}, \Delta r\_Pm1_{out}, \Delta f\_V1_{in}, \Delta r\_V1_{in}, \Delta f\_V1_{out}, \Delta r\_V1_{out}\}$ *is not a MSS for the requirement* $\{Pm1(up), P1(lk)\}$*; indeed, a possible MSS* O^* *consists in measuring just* $\{\Delta f\_Pm1_{out}, \Delta r\_Pm1_{out}\}$*, i.e. the variables of O that concern the output of pump* $Pm1$.

Discriminability Relations

As stated above, in some cases one could be interested in knowing whether a given observability O allows one to discriminate between two behavioral modes of two components (i.e. $c_1(bm)$ and $c_2(bm')$ are discriminable w.r.t. O).

However, in most cases it is much more interesting to know the set of observabilities which guarantee that a set of discriminability requirements are satisfied.

We could have cases where the discriminability requirements are not satisfied under any observability O. In many cases there are many of such observabilities and then we could be interested in selecting in a flexible way the most appropriate. The notion of MSS introduced above is a direct way for expressing preferences in terms on the cardinality of sensor sets.

First of all we need a way for easily representing and manipulating the set of all observabilities. For this reason we first introduce the notion of *observation switches*.

Definition 6 *Given an endogenous port* $p \in P_{end}$ *its associated* observation switch sw_p *is a variable with* $dom(sw_p) = \{yes, no\}$. *We denote as* SW *the set of switches associated with variables in* P_{end}.

An *observation switch* sw_p represents the fact that endogenous port p is currently observable (value *yes*) or not (value *no*). Therefore, an instance $\mathcal{Y}_{SW}$ of the SW variables identifies a degree of observablity $O_{\mathcal{Y}_{SW}} \subseteq P_{end}$.

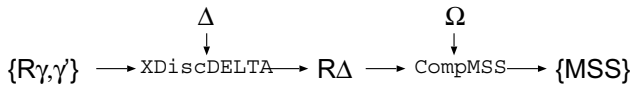


Figure 4: Scheme of the Computation of MSSs.

We can now introduce the notion of *parametric* discriminability relation between two pairs $(c_1(bm_i), c_2(bm_j))$ and $(c_1(bm_k), c_2(bm_l))$ of behavioral modes of components c_1 and c_2. Such a relation aims at representing all the possible degrees of system observability which make $(c_1(bm_i), c_2(bm_j))$ discriminable from $(c_1(bm_k), c_2(bm_l))$ according to Definition 2.

The following definition formalizes such a notion by exploiting the mechanism of *observation switches*.

Definition 7 *Given $\gamma = (c_1(bm_i), c_2(bm_j))$ and $\gamma' = (c_1(bm_k), c_2(bm_l))$, the* parametric discriminability relation *(PDR) $\mathcal{R}_{\gamma,\gamma'}$ is a relation whose tuples are instantiations $\mathcal{Y}_{SW}$ of SW variables.*
A tuple $(\mathcal{Y}_{SW})$ is included in the relation $\mathcal{R}_{\gamma,\gamma'}$ iff γ and γ' are discriminable w.r.t. $O_{\mathcal{Y}_{SW}}$.

Example 5 *In the hydraulic system, we associate a switch sw_p to each endogenous port $p \in P_{end}$, namely each variable representing the qualitative deviation of the flow or pressure at one of the connection points (see Figure 1); so we have 18 switches.*
According to Example 3, if $\delta = (Pm1(up), P1(lk))$, we will evaluate relation $\mathcal{R}_{\gamma,\gamma'}$ for many pairs γ, γ'. In case $\gamma = (Pm1(up), P1(ok))$ and $\gamma' = (Pm1(ok), P1(lk))$, $\mathcal{R}_{\gamma,\gamma'}$ will contain a tuple where:
$sw_{\Delta f\_Pm1_{out}}$, $sw_{\Delta r\_Pm1_{out}}$, $sw_{\Delta f\_V1_{in}}$, $sw_{\Delta r\_V1_{in}}$, $sw_{\Delta f\_V1_{out}}$ *and* $sw_{\Delta r\_V1_{out}}$
have value yes *while the remaining switches have value* no.
However, relation $\mathcal{R}_{\gamma,\gamma'}$ contains many other tuples, including the one corresponding to the MSS O^* of Example 4, where just $sw_{\Delta f\_Pm1_{out}}$ and $sw_{\Delta r\_Pm1_{out}}$ have value yes.

A Sketch of the Computational Process

Before describing in detail how the PDRs $\mathcal{R}_{\gamma,\gamma'}$ are computed it is worth explaining the overall computation of MSSs in our approach (Figure 4).

The starting point is the set of discriminability requirements $\Delta = \{\delta_1, \ldots, \delta_m\}$ the user is interested in.

Given a specific discriminability requirement $\delta_i = (c_1(bm), c_2(bm'))$, the system has to compute the (parametric) discriminability relation $\mathcal{R}_{\delta_i}$ for $c_1(bm)$ and $c_2(bm')$ starting from the PDRs $\mathcal{R}_{\gamma,\gamma'}$, since the discriminability of $c_1(bm)$ and $c_2(bm')$ is defined in terms of pairs of assignments of behavioral modes to c_1 and c_2 (see Definition 3).

On the basis of discriminability relations $\mathcal{R}_{\delta_i}$ it is possible to compute the global discriminability relation $\mathcal{R}_\Delta$ representing the set of all possible degrees of system observability able to satisfy all the discrimination requirements in Δ.

At this point the user can go on in the analysis by requiring the system to compute the Minimal Sensor Sets. Actually in our architecture the approach is even more flexible

AugmentDT(SD)
1 $\hat{DT} = DT$, $SW = \emptyset$, $RD = \emptyset$
2 **foreach** $o \in O$
3 $SW_O = SW \cup \{sw_o\}$, $RD = RD \cup \{rd_o\}$
4 $dom(sw_o) = \{yes, no\}$, $dom(rd_o) = dom(o) \cup \{abs\}$
5 $DT_{sw_o} = \emptyset$
6 **foreach** $v \in dom(o)$
7 $DT_{sw_o} = DT_{sw_o} \cup \{(sw_o(yes), o(v), rd_o(v)\}$
8 $DT_{sw_o} = DT_{sw_o} \cup \{(sw_o(no), o(v), rd_o(abs)\}$
9 $\hat{DT} = \hat{DT} \bowtie DT_{sw_o}$

Figure 5: Augmenting *DT* with observability switches.

since the user can specify a set of constraints Ω on the sensors. In fact, the user may specify that an observable o is available in any case (by expressing a constraint in Ω that assigns the value yes to sw_o), or that o is not to be considered (in this case it is sufficient to constrain $sw_{o'}$ to take the value no).

By taking into consideration Ω and the *discriminability relation* $\mathcal{R}_\Delta$ the module `CompMSS()` is able to compute all the Minimum Sensor Sets and therefore is able to provide the user with the minimum sets of sensors that satisfy his/her discrimination requirements. In this computation, `CompMSS()` exploits another set of precomputed relations CSS_i that we'll describe in section .

It is worth noting that the discriminability relations $\mathcal{R}_{\gamma,\gamma'}$ do not directly depend on the discrimination requirement Δ and on the constraints on sensors Ω, but just on the domain model. For this reason in our architecture $\mathcal{R}_{\gamma,\gamma'}$ are computed just once, stored in a repository (called PDR memory) and used as many times as it is necessary for computing the $\mathcal{R}_\Delta$ any time a user provides a Δ. As we will see in the section devoted to experimental analysis such a form of precompilation and reuse plays a significant role in dramatically reducing the computational cost.

In the following section we will describe how to compute $\mathcal{R}_{\gamma,\gamma'}$ and $\mathcal{R}_\Delta$ while the computation of MSSs is described in a later section.

Computing PDRs

Before computing PDRs $\mathcal{R}_{\gamma,\gamma'}$, it is necessary to perform a step for augmenting the Domain Theory in order to add the observation switches (with their models).

This step is performed by function `AugmentDT` (Figure 5) which returns the augmented Domain Theory $\hat{DT}$

Following Definition 6, function `AugmentDT` introduces, for each endogenous variable o, a switch sw_o that can take values *yes* or *no* (lines 3-4).
In order to capture the influence of switch sw_o on what we actually observe, a variable rd_o is also introduced; such a variable is intended to take the value of o when sw_o is set to *yes*, and a value abs (for absent) when sw_o is set to *no*. Lines 5-8 construct relation DT_{sw_o}, i.e. the portion of the model associated with sw_o, according to this goal. Finally, relation DT_{sw_o} is added to the Domain Theory in line 9.

The **computation of** $\mathcal{R}_{\gamma,\gamma'}$ where $\gamma = (c_1(bm_i), c_2(bm_j))$

PDRGamma($(c_1(bm_i), c_2(bm_j))$**,** $(c_1(bm_k), c_2(bm_l))$**)**
1 $\mathcal{V} = P_{exo} \cup C \cup SW \cup RD$
2 $\hat{DT}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\hat{DT})$
3 $\hat{DT}_{i,j} = \sigma_{c_1(bm_i),c_2(bm_j)}(\hat{DT}_{\mathcal{V}})$
4 $\hat{DT}_{i,j} = \Pi_{P_{exo} \cup SW \cup RD}(\hat{DT}_{i,j})$
5 $\hat{DT}_{k,l} = \sigma_{c_1(bm_k),c_2(bm_l)}(\hat{DT}_{\mathcal{V}})$
6 $\hat{DT}_{k,l} = \Pi_{P_{exo} \cup SW \cup RD}(\hat{DT}_{k,l})$
7 $\hat{DT}_{com} = \hat{DT}_{i,j} \cap \hat{DT}_{k,l}$
8 $\hat{DT}_{com} = \Pi_{P_{exo} \cup SW}(\hat{DT}_{com})$
9 $\hat{DT}_{diff} = dom(P_{exo} \cup SW) \backslash \hat{DT}_{com}$
10 $\hat{DT}_{diff} = \Pi_{SW}(\hat{DT}_{diff})$

Figure 6: Computation of $\mathcal{R}_{\gamma,\gamma'}$.

and $\gamma' = (c_1(bm_k), c_2(bm_l))$ is performed by the function `PDRGamma()` reported in Figure 6; such a function determines which are the observabilities which make $(c_1(bm_i), c_2(bm_j))$ discriminable from $(c_1(bm_k), c_2(bm_l))$ according to Definition 2.

For efficiency reasons that will be clearer when we briefly discuss the implementation based on OBDDs, the algorithm completely avoids the explicit enumeration of the tuples of the relations involved in the computation, but directly works on the relations. In particular, lines 1-2 take care of forgetting the internal variables I from relation $\hat{DT}$ as well as the endogenous ports P_{end} since the variables that must be taken into consideration to verify discriminability are the observables RD added in the previous step. The forgetting is accomplished by projecting $\hat{DT}$ on the other variables, namely $P_{exo} \cup C \cup SW \cup RD$. In lines 3 and 4 the algorithm computes $\hat{DT}_{i,j}$ by restricting the domain theory to the case where $c_1(bm_i)$ and $c_2(bm_j)$ and then projects such a relation on $P_{exo} \cup SW \cup RD$ by discarding all the variables referring to the components in C (note that c1 and c2 have been restricted to take a specific behavioral mode $c_1(bm_i)$ and $c_2(bm_j)$ respectively). It is worth noting that this is an efficient way for implementing the quantification over all the assignments $\mathcal{C}_{\overline{1,2}}$ of $C \backslash \{c1, c2\}$ required by Definition 2.

An analogous remark applies to the computation of relation $\hat{DT}_{k,l}$ where the behavioral modes $c_1(bm_k)$ and $c_2(bm_l)$ are asserted (lines 5 and 6).

Relation $\hat{DT}_{com}$ (line 7) is obtained by intersecting $\hat{DT}_{i,j}$ and $\hat{DT}_{k,l}$ and therefore contains instances of $P_{exo} \cup SW$ that could be responsible for the indiscriminability between $(c_1(bm_i), c_2(bm_j))$ and $(c_1(bm_k), c_2(bm_l))$ by sharing common observations; after this step, the values of RD variables can be safely forgotten (line 8).

The last steps consist in taking the complement $\hat{DT}_{diff}$ of $\hat{DT}_{com}$ (which contains the instances of $P_{exo} \cup SW$ that necessarily discriminate between $(c_1(bm_i), c_2(bm_j))$ and $(c_1(bm_k), c_2(bm_l))$ in terms of SR values) (line 9) and, finally, in disregarding the P_{exo} variables so that the resulting relation $\mathcal{R}_{\gamma,\gamma'}$ is defined just in terms of SW variables (line 10).

The **computation of** $\mathcal{R}_{\Delta}$ is activated when the user

PDRDelta ($\Delta = \{\delta_1, \ldots, \delta_m\}$**)**
1 $\mathcal{R}_{\Delta}$ = dom(SW)
2 **foreach** $(\delta_i = (c_1(bm), c_2(bm'))$
3 $\mathcal{R}_{\delta_i}$ = dom(SW)
4 **foreach** $bm_i, bm_k \in dom(c_1)$
5 **foreach** $bm_j, bm_l \in dom(c_2)$
6 $\gamma = (c_1(bm_i), c_2(bm_j))$
7 $\gamma' = (c_1(bm_k), c_2(bm_l))$
8 **if** (bm_i == bm $\wedge$ bm_j == bm') **then** case := 1
9 **if** (bm_i == bm $\wedge$ bm_j != bm') **then** case := 2
10 **if** (bm_i != bm $\wedge$ bm_j == bm') **then** case := 3
11 **if** (case == 1 $\wedge$ $\gamma == \gamma'$) **then** *skip*
12 **if** (case == 2 $\wedge$ bm_k == bm)) **then** *skip*
13 **if** (case == 3 $\wedge$ bm_l == bm')) **then** *skip*
14 **if** ($\mathcal{R}_{\gamma,\gamma'}$ is not in PDR memory) **then**
15 $\mathcal{R}_{\gamma,\gamma'}$ = PDRGamma(γ, γ')
16 add $\mathcal{R}_{\gamma,\gamma'}$ to PDR memory
17 $\mathcal{R}_{\delta_i} = \mathcal{R}_{\delta_i} \cap \mathcal{R}_{\gamma,\gamma'}$
18 $\mathcal{R}_{\Delta} = \mathcal{R}_{\Delta} \cap \mathcal{R}_{\delta_i}$

Figure 7: Computation of $\mathcal{R}_{\Delta}$.

provides a set of discrimination requirements $\Delta = \{\delta_1, \ldots, \delta_m\}$. Such a computation is performed by the function `PDRDelta()` (Figure 7) which examines in turn all the discriminability requirements δ_i in Δ.

Given a specific $\delta_i = (c_1(bm), c_2(bm'))$, the system has to compute the discriminability relation $\mathcal{R}_{\delta_i}$ for $c_1(bm)$ and $c_2(bm')$ starting from the discriminability relations $\mathcal{R}_{\gamma,\gamma'}$ since the discriminability of $c_1(bm)$ and $c_2(bm')$ is defined in terms of pairs of assignments of behavioral modes to c_1 and c_2 according to Definition 3.

The computation of the relation $\mathcal{R}_{\delta_i}$ for a discrimination requirement $\delta_i = (c_1(bm), c_2(bm'))$ is indeed a direct implementation of Definition 3 where $\gamma = (c_1(bm_i), c_2(bm_j))$ and $\gamma' = (c_1(bm_k), c_2(bm_l))$ obey of the three cases considered in the definition.

The function `PDRGamma()` is invoked for the appropriate pair of γ and γ' and it returns $\mathcal{R}_{\gamma,\gamma'}$ which is used for iteratively computing the relation $\mathcal{R}_{\delta}$. It is worth looking to lines 14-16 of the algorithm, where a check is performed to verify whether $\mathcal{R}_{\gamma,\gamma'}$ has been already computed and saved in the PDR memory. Just in case $\mathcal{R}_{\gamma,\gamma'}$ is not present, `PDRGamma()` is invoked and the result is saved in the PDR memory. In this way it is apparent that the computation of $\mathcal{R}_{\gamma,\gamma'}$ is performed just once and reused many times.

In many cases, it could be useful to perform off line the computation of $\mathcal{R}_{\gamma,\gamma'}$ for all possible combinations of $\gamma = (c_1(bm_i), c_2(bm_j))$ and $\gamma' = (c_1(bm_k), c_2(bm_l))$ for all components, so that for any possible set Δ of discrimination requirements the function `PDRDelta()` has never to invoke `PDRGamma()` but has just to retrieve the appropriate $\mathcal{R}_{\gamma,\gamma'}$ from the PDR memory.

Computing MSS from PDRs

The final step consists in computing the Minimum Sensor Sets by exploiting the result of `PDRDelta()` i.e. $\mathcal{R}_{\Delta}$.

CardSensorSets(P_{end})
1 m := $|P_{end}|$
2 $CSS_0 = (sw_{o_1}(no), \ldots, sw_{o_m}(no))$
3 **for** i=1 **to** m
4 $CSS_i = \emptyset$
5 **for** j=1 **to** m
6 $CSS_{i,j} = \sigma_{sw_{o_j}(no)}(CSS_{i-1})$
7 $CSS_{i,j} = \Pi_{SW \setminus \{sw_{o_j}\}} CSS_{i,j}$
8 $CSS_{i,j} = CSS_{i,j} \bowtie \{(sw_{o_j}(yes))\}$
9 $CSS_i = CSS_i \cup CSS_{i,j}$

Figure 8: Computation of CSS_i, $i = 0, \ldots, |P_{end}|$.

In principle such a minimization step could be very expensive from a computational point of view but we can adapt the efficient approach we have developed for the computation of minimum cardinality diagnoses (Torasso & Torta 2006) to the computation of minimum sensor sets.

The basic idea is to precompute a set of filters CSS_i, where CSS_i lists all the possible combinations of switches sw_{o_j} with exactly i switches assuming the value yes. In other words each CSS_i represents all the possible observabilities that involve exactly i observable variables. Therefore CSS_0 represents the case where nothing is observable (all the switches sw_{o_j} have value no) while CSS_m represents the case when all the m variables in P_{end} are actually observed (all the switches sw_{o_j} have value yes). The computation of CSS_i is performed by the function `CardSensorSets()` (Figure 8) which iteratively computes each CSS_i starting from CSS_0 (see (Torasso & Torta 2006) for details).

The actual computation of MSSs is performed by the function `CompMSS()` (Figure 9). First of all the algorithm takes into consideration the set of constraints Ω on the sensors provided by the user. Note that in many cases the user may not be willing or able to provide constraints on the availability (or not) of specific sensor, so in these cases the relation Ω is unconstrained (i.e. $\Omega = dom(SW)$).

The relation $\mathcal{R}_{\Delta,\Omega}$ contains now all the observabilities satisfying the discrimination requirements in Δ compatible with the constraints in Ω. The minimization step is a simple task once one has at disposal relations CSS_i: it is sufficient to verify whether the intersection of $\mathcal{R}_{\Delta,\Omega}$ with CSS_i is not empty starting from CSS_0. As soon as we find a non-empty intersection for index i, relation *MSS* represents the set of all the possible combinations of i sensors which satisfy both the discriminability requirements and the constraints on the sensors.

Example 6 *Let us consider again the hydraulic system and let us suppose that Δ contains the two discriminability requirements already mentioned in Example 1, namely $\{(Pm1(up), P1(lk)), (V1(sc), V2(sc))\}$.*
In case Ω is unconstrained, the computation of MSS *shows that at least 6 sensors are needed in order to satisfy Δ; the algorithm computes four different minimal sensor sets among which the following one, where:*

$sw_{\Delta r\_Pm1_{out}}$, $sw_{\Delta r\_V1_{in}}$, $sw_{\Delta r\_V1_{out}}$,
$sw_{\Delta f\_J_{in1}}$, $sw_{\Delta r\_V2_{in}}$, $sw_{\Delta r\_V2_{out}}$
have value yes *while the remaining switches have value* no.

In case the discriminability requirements are the same as above but the user has provided some constraints about observability, the algorithm can return a different solution. In particular, if Ω requires that Δf_J_{out} and Δr_J_{out} are observable (i.e. their switches are set to yes*) and Δf_J_{in1} and Δf_J_{in2} are unobservable (i.e. their switches are set to* no*) the algorithm computes six different minimal sensor sets containing 8 sensors each (the two sensors required by Ω plus six other sensors); one of these optimal solutions requires that the following six additional switches:*

$sw_{\Delta r\_Pm1_{out}}$, $sw_{\Delta r\_V1_{in}}$, $sw_{\Delta r\_V1_{out}}$,
$sw_{\Delta f\_V2_{out}}$, $sw_{\Delta r\_V2_{in}}$, $sw_{\Delta r\_V2_{out}}$
have value yes *while the remaining switches have value* no.

CompMSS($\mathcal{R}_\Delta$, Ω)
1 $\mathcal{R}_{\Delta,\Omega} = \mathcal{R}_\Delta \cap \Omega$
2 i = 0
3 $MSS = \mathcal{R}_{\Delta,\Omega} \cap CSS_i$
4 **while** $MSS == \emptyset \wedge i < |O|$
5 i = i + 1
6 $MSS = \mathcal{R}_{\Delta,\Omega} \cap CSS_i$

Figure 9: Computation of Set *MSS*.

An Implementation Based on OBDDs

The algorithms reported in the previous section are given in terms of relational operators. A close look at the algorithms shows that in all cases just a relatively small number of relational operations is required for computing $\mathcal{R}_{\gamma,\gamma'}$, $\mathcal{R}_\Delta$ and *MSS*.

However, the critical issue is the size of the relations which constitute the inputs and (intermediate) results of the algorithms.
Since such sizes can in general be huge, we have adopted OBDDs for encoding and manipulating all the relations involved in the algorithms (including the Domain Theory *DT*). The choice of OBDDs has been dictated by three motivations: first of all, OBDDs are well known for being able to compactly represent huge relations in many practical cases (e.g. (Bertoli *et al.* 2001), (Jensen & Veloso 1999), (Torta & Torasso 2006a)); secondly, it is straightforward to map all the relational operations of our algorithms into operations that manipulate OBDDs, and most operations are guaranteed to have polynomial complexity w.r.t. the OBDDs they operate upon (Bryant 1992); finally, public domain libraries that support construction and manipulation of OBDDs are available (e.g. the *Buddy* package).

In the section devoted to experimental analysis we will see that the adoption of OBDDs allows to encode in a very compact way the relations used during the computation of MSSs for a non-trivial system and that the computation is performed in an efficient way form a computational point of view.

For the computation of MSSs we have also a theoretical result which guarantees that the potentially very expensive

R	#relations	size avg	size max
$\mathcal{R}_{\gamma,\gamma'}$	2880	44	752
$\mathcal{R}_{\delta}$	360	420	1860
$\mathcal{R}_{\Delta}$	10	196.6	616
MSS	10	77	185
MSS_{Ω}	10	74.3	102

Table 1: Sizes of OBDDs Representing Relevant Relations.

R	#relations	time avg	time max
$\mathcal{R}_{\gamma,\gamma'}$	2880	1.05	160
$\mathcal{R}_{\delta}$	360	0.61	20
$\mathcal{R}_{\Delta}$	10	38.1	70
MSS	10	0	0
MSS_{Ω}	10	0	0

Table 2: Times for Computing Relevant Relations (msec).

task of computing the minimal sensor set can be done in polynomial time with respect to the size of the OBDD $\mathcal{O}_{\mathcal{R}_\Delta}$ encoding $\mathcal{R}_\Delta$.

Property 1 *Let $\mathcal{O}_{\mathcal{R}_\Delta}$ be an OBDD encoding relation $\mathcal{R}_\Delta$; then, OBDD $\mathcal{O}_{MSS}$ encoding relation MSS can be computed by* `CompMSS()` *in time $O(|P_{end}|^3 \cdot |\mathcal{O}_{\mathcal{R}_\Delta}|)$.*

This property mirrors a similar result obtained for Minimum Cardinality Diagnoses whose proof is reported in (Torasso & Torta 2006). When the OBDD encoding $\mathcal{O}_{\mathcal{R}_\Delta}$ is small, we have the guarantee that `CompMSS()` can be executed efficiently.

Experimental Results

We have tested the approach in two different domains: the hydraulic system used as a running example and the combinatorial digital circuit c74182 taken from the ISCAS85 repository. In this section we report results obtained with c74182, which represents a more challenging test bed.

The c74182 circuit is a 4 bit carry look-ahead with 9 input signals and 5 outputs. In the models of the circuits reported in ISCAS85, different types of faults are considered: each logical gate can be in a *ok*, *sa0* (stuck at 0) or *sa1* (stuck at 1) mode. Moreover also the connections can be faulty so that the total number of components (logical gates plus connections) becomes 70. We have considered as possible observation measurements the output of all the logical gates, so that we have 28 switches for representing all the possible observability degrees (included 5 switches for the 5 system outputs).

The direct application of the notion of discriminability of Definition 3 does not produce significant result because almost no pair of behavioral modes is discriminable under the requirements that all other components may be in any mode. For this reason we are reporting results concerning an analysis of discriminability requirements where at most one logical gate is faulty.

For collecting the statistics about $\mathcal{R}_{\gamma,\gamma'}$ and $\mathcal{R}_\delta$, we have considered the computation of discriminability for **360** different $\mathcal{R}_\delta$ from **2880** different $\mathcal{R}_{\gamma,\gamma'}$.

For collecting the statistics about $\mathcal{R}_\Delta$, we have considered the computation of discriminability for **10** different Δs, each of which requiring the computation of the discriminability for **90** different δs from **720** different (γ, γ').

The experimental results are reported in Table 1 and in Table 2.

As concerns the computation of $\mathcal{R}_{\gamma,\gamma'}$ we have started from the encoding of the domain theory via OBDDs. By using a simple depth-first heuristic for variable ordering, the system produces an encoding of the DT of **5496** nodes (obtained from an OBDD of **31942** nodes by removing the non-observable endogenous variables). The next step is the application of `AugmentDT` which produces an encoding of $\hat{DT}$ of **8781** nodes. Therefore, the augmentation of the domain theory for taking into consideration the switches is far from dramatic.

The first positive result reported in Table 1 is the extremely compact encoding of the $\mathcal{R}_{\gamma,\gamma'}$ relations: in fact the average size of their OBDD encoding is just **44** nodes and also the maximum size is low (**752** nodes). This makes possible to actually store the results into the PDR memory.

Another good result concerns the cost of computing each $\mathcal{R}_{\gamma,\gamma'}$ in terms of CPU time (see Table 2). The computation of a single $\mathcal{R}_{\gamma,\gamma'}$ can be done quite efficiently: on average, **1.05** msec with a maximum time of **162** msec.

We exploit the capability of the system to compute just once each $\mathcal{R}_{\gamma,\gamma'}$ and to store the result into the PDR memory. Tables 1 and 1 show that, assuming that all the needed $\mathcal{R}_{\gamma,\gamma'}$ are in the PDR memory, the computations of $\mathcal{R}_\delta$ and of $\mathcal{R}_\Delta$ can be done quite efficiently both in terms of space (the size of the OBDDs encoding $\mathcal{R}_\delta$ and $\mathcal{R}_\Delta$ is always under control) and in terms of CPU time.

It is worth noting that the efficiency in time of the computation of $\mathcal{R}_\delta$ and of $\mathcal{R}_\Delta$ depends essentially on the reuse of the $\mathcal{R}_{\gamma,\gamma'}$. In particular, as reported in Table 2, the average time required to compute $\mathcal{R}_\Delta$ turned out to be **38.1** msec while the average time to compute the **720** $\mathcal{R}_{\gamma,\gamma'}$ necessary for "assembling" $\mathcal{R}_\Delta$ was **920.6** msec; thanks to the precompilation of $\mathcal{R}_{\gamma,\gamma'}$ we saved almost **96%** of the time needed without precompilation.

Once the $\mathcal{R}_\Delta$ has been computed the step of computing MSS can be done in a very efficient way both in time and space. The size of the OBDDs encoding MSS is small (**77** nodes on average). In all the ten considered cases we had that the time required for finding the minimum sensor set was under the threshold of 1 millisecond.

We further tested the performance of `CompMSS` by running it in presence of constraints on the sensor sets. In particular we have included in Ω all the 5 system outputs. The results for the computation of MSS_Ω are almost identical to the case with no constraints.

Conclusions

In the present paper we have proposed and discussed a novel method for computing minimal sensor sets given a desired level of diagnosability and positive/negative observability constraints.

Although the problem of computing the Minimal Sensor

Sets has been previously addressed in the literature (e.g. (Scarl 1994), (Travé-Massuyès, Escobet, & Milne 2001), (Krysander & Nyberg 2002), (Travé-Massuyès, Escobet, & Olive 2006)), the presented approach is novel in that it gets a high level of flexibility and performance through the precompilation of parametric discriminability relations which can be efficiently computed and parsimoniously stored using Ordered Binary Decision Diagrams as a symbolic representation.

The experimental results confirm the effectiveness of the approach in a non-trivial domain.

The paper shares some assumptions, concepts and technical solutions with (Torta & Torasso 2006b) and (Esser & Struss 2007); however, the aim is significantly different (although related), since (Torta & Torasso 2006b) addresses abstraction and (Esser & Struss 2007) addresses testing.

In the present paper we have used minimum cardinality of sensors as a preference criterion, but other criteria exist based on different notions of the *cost* of a sensor set; in particular, an obvious generalization would be to allow different costs for the sensors, so that the cumulative cost of a sensor set is not just its cardinality.
We believe that the approach described in this paper could be generalized to cover different costs, provided there is a limited number of (qualitative) possible costs for each sensor; in such a case, indeed, it would be possible to compute relations (corresponding to the CSS relations in this paper) that represent all the possible combinations of sensors whose cumulative cost is equal to a given level.

References

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. IJCAI*, 473–478.

Bryant, R. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24:293–318.

Esser, M., and Struss, P. 2007. Fault-model-based test generation for embedded software. In *Proc. IJCAI*, 342–347.

Jensen, R. M., and Veloso, M. M. 1999. Obdd-based universal planning: Specifying and solving planning problems for synchronized agents in nondeterministic domains. *LNCS* 1600:213–248.

Krysander, M., and Nyberg, M. 2002. Structural analysis for fault diagnosis of dae systems utilizing mss sets. In *Proc. IFAC World Congress*.

Scarl, E. 1994. Sensor placement for diagnosability. *Annals of Mathematics and Artificial Intelligence* 1(1–4):493–509.

Struss, P.; Malik, A.; and Sachenbacher, M. 1996. Qualitative modeling is the key to automated diagnosis. In *Proc. IFAC96*.

Torasso, P., and Torta, G. 2006. Model-based diagnosis through obdd compilation: a complexity analysis. *Lecture Notes in Computer Science* 4155:287–305.

Torta, G., and Torasso, P. 2006a. On the use of obdds in model-based diagnosis: an approach based on the partition of the model. *Knowledge Based Systems* 19(5):316–323.

Torta, G., and Torasso, P. 2006b. Qualitative domain abstractions for time-varying systems:
an approach based on reusable abstraction fragments. In *Proc. DX*, 265–272.

Travé-Massuyès, L.; Escobet, T.; and Milne, R. 2001. Model-based diagnosability and sensor placement. application to a frame 6 gas turbine sub-system. In *Proc.* 17^{th} *Int. Joint Conference on Artificial Intelligence - IJCAI01*, 551–556.

Travé-Massuyès, L.; Escobet, T.; and Olive, X. 2006. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man and Cybernetics PART A* 36(6):1146–1160.

Ontologies for Data Mining and Knowledge Discovery to Support Diagnostic Maturation

Timothy J. Wilmering
The Boeing Company
P.O. Box 516 M/C S106 3075
St. Louis, MO. 63166
timothy.j.wilmering@boeing.com

John W. Sheppard
The Johns Hopkins University
3400 N Charles Street
Baltimore, MD 21218
jsheppa2@jhu.edu

Abstract

Initial test and maintenance solutions that are deployed to support new complex systems are generally imperfect and are initially liable to contribute substantially to system ownership costs. This is because development of effective health management solutions requires prediction of complex systemic interactions and the effect of presupposed external stimuli. It is nearly always the case that unforeseen emergent behavior (those that result from unpredicted system interactions) of fielded systems within their operational context create deviations from anticipated health management system performance. This suggests a need for processes and tools to monitor the effectiveness of product health management solutions in their application domains, collect data that validates and documents system performance, and pinpoint and analyze relevant patterns that can help mitigate the issues that arise. The process of identifying and implementing corrective actions as required to satisfy customer support requirements is known as diagnostic maturation and often draws on techniques from data mining and knowledge discovery.

Most current approaches to data mining involve analyzing low-level data elements to attempt to induce previously unknown relationships among those data elements. Techniques such as clustering, association rule learning, feature extraction, and classification pervade the data mining literature. Unfortunately, most data mining implementations are either "blind" in that they consider all available data, or they depend on a human expert to identify the types of relationships of interest. Recent developments in defining ontologies for a domain provide significant potential in data mining in general and diagnostic maturation in particular. In this paper, we discuss a framework and approach for utilizing health management ontologies to guide the maturation process and better streamline automatic (or semi-automatic) knowledge discovery to improve diagnostics.

Diagnostic Maturation Process

Initial test and maintenance solutions that are deployed to support new complex systems are generally imperfect and are initially liable to contribute substantially to system ownership costs. This is because development of effective health management solutions requires prediction of complex systemic interactions and the effect of presupposed external stimuli. It is nearly always the case that unforeseen emergent behavior (those that result from unpredicted system interactions) of fielded systems within their operational context create deviations from anticipated health management system performance. This suggests a need for processes and tools to monitor the effectiveness of product health management solutions in their application domains, collect data that validates and documents system performance, and pinpoint and analyze relevant patterns that can help mitigate the issues that arise. The ability to mature the effectiveness of fielded system test, diagnostic, and maintenance procedures is a critical factor in an overall system support posture. The process of identifying and implementing corrective actions as required to satisfy customer support requirements is known as diagnostic maturation.

Recognizing that data mining in general and diagnostic maturation in particular are difficult, the purpose of this paper is to discuss one approach to streamlining the process by using domain ontologies. Specifically, in this paper, we discuss a framework and approach (one of many possible) for utilizing health management ontologies to guide the maturation process and better streamline automatic (or semi-automatic) knowledge discovery to improve diagnostics. The focus is on using the ontology to focus data mining and reduce the size of the search space to only those portions directly relevant to a specific maturation question.

The initial frameworks to support health management system maturation are put in place in the conceptual design stage and should be developed throughout the system life cycle. Model-based approaches (e.g., logic models, dependency models, qualitative models, and physics-based

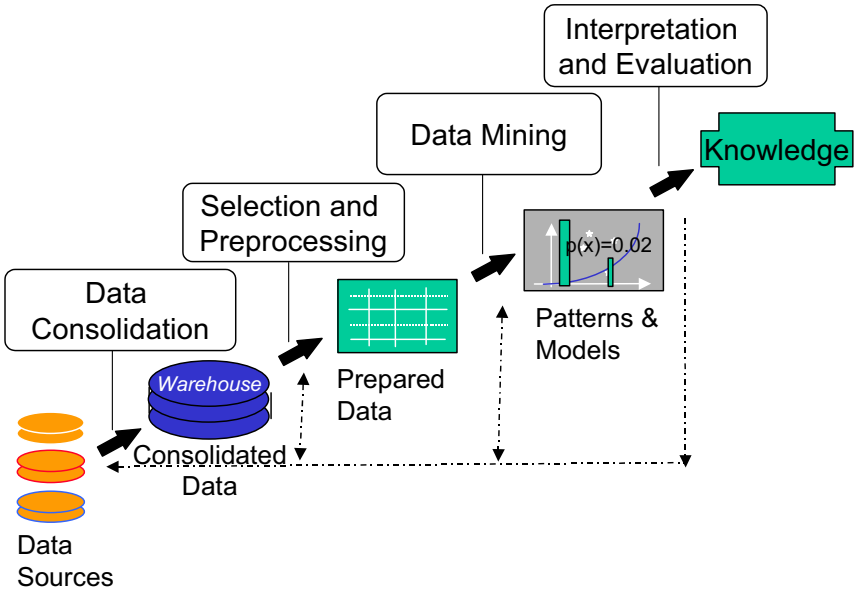


Figure 1. Traditional Knowledge Discovery Process (Han and Kamber, 2006)

models) provide excellent representational tools for developing and documenting system design choices that support traceability of design decisions and can be refactored during the corrective action analysis and development process. When unexpected and unplanned system level design interactions, operational and environmental stresses, or other influences create a system readiness issue or cost of ownership problem, the models will be in place to support the remedial actions that must be taken. This represents an iterative closed loop process of root cause analysis, corrective action deployment and reevaluation called a Maturation Cycle, or more formally in some circles, the FRACAS (Failure Reporting and Corrective Action System) process. In either case, the goal is to determine a corrective action that prevents or minimizes recurrence of the reported problem in subsequent use of the product (Wilmering, 2001).

Maturation is hard because of two fundamental issues (Wilmering, 2003):

1. While it is generally perceived that data collection is a prerequisite for the maturation process, data collection is in fact a difficult issue. The product data that is typically required for maturation analysis is generally stored in disparate heterogeneous data systems - this makes access, retrieval, and integration of the requisite information a costly and often incomplete process at best.
2. There are several categories of analysis required to support maturation: correlation analysis (identifying the problem and correlating the data which supports and characterizes that identification), root cause analysis, and analysis of the support system that provides a framework for the processes in question and corrective action. At issue is the fact that the data required to perform these analyses is scattered among many different data sources and systems – pulling it all together can be challenging, to say the least.

Knowledge Discovery

We propose that formal Knowledge Discovery techniques may offer significant benefit to the diagnostic maturation process. Suppose a Maturation Cycle is triggered by some event or series of events during the operation of a system. Perhaps a repeated test failure is determined to be unfounded – this may first be noticed by repeated False Alarms or Cannot Duplicates involving a system component. An analyst may first characterize the problem based on the information as it is initially presented: what is the part whose failure is misdiagnosed, what information is already available that helps to characterize the problem (times, locations, maintenance scenarios, etc). Once the initial problem statement is formalized, the Knowledge Discovery (KD) process is described in Han and Kamber (2006) can begin as follows (Figure 1):

Data collection and consolidation: Identify relevant data and factors, find relevant data sources, locate data in sources and formulate queries.

Prepare Data (Data Integration): Develop data correlation keys, retrieve data, integrate data, clean data, examine data.

Data Mining: Data mining is the heart of the KD process. Select the appropriate methods for pattern extraction from the large data sets collected in the previous step, then apply any of numerous classification and pattern matching techniques to extract relevant relationships from the data sets.

Interpretation and Evaluation (Analysis of results): Interpretation of results may employ further data reduction mechanisms, use of visualization techniques, or other

methods to enhance presentation and interpretation of the data for human consumption.

This may be an iterative process as patterns emerge, facts are discovered, or experiments are designed and carried out.

Knowledge discovery is hard for two fundamental reasons:

1. There is a wide variety of available tools, techniques, and representations for data that must be considered throughout the knowledge discovery process.
2. Given the typically large amount of data that must be mined, the process of finding interesting relationships in a combinatorially large data space is itself computationally hard.

Given the range of possible tools and techniques available, as well as the range of possible combinations of data elements to be considered for analysis, the problem of knowledge discovery in general (and data mining in particular) is difficult. The focus in this paper is providing an approach to reduce the scope of the search required in knowledge discovery by using ontologies. Currently, the common approaches to data mining include (among others).

- On Line Analytical Processing (OLAP)—A multi-dimensional analysis technique involving the aggregation of "cubes" (actually, hypercubes) of data to determine relationships among the data. The aggregation process is analogous to marginalization in probability theory, except counts are returned rather than probabilities.
- Statistical Analysis—A collection of tools range from correlation analysis, to trending, to regression where models of data are constructed uses traditional tools from statistics. In fact, all of the techniques used in data mining can (and often do) benefit from applying combinations of statistical tools.
- Cluster Analysis—An unsupervised learning technique where the data is grouped by common attributes, often based on a distance function. Clustering can be used to identify a set of candidate classes for the data that might not have been known previously.
- Association Rule Analysis—An unsupervised learning technique where correlations between data elements are examined in an attempt to extract logical relationships between those attributes. Learning association rules is similar to learning decision rules using supervised learning techniques such as FOIL (Quinlan, 1990).
- Pattern Classification—A class of algorithms, often considered under supervised learning, where data with associated classes or labels are used to derive a compact model of those classes for future classification of new data.

As we see, determining which of these tools to apply can, itself, be a significant challenge. Interesting work at NYU (Bernstein, Hill, and Provost, 2002) is applying an ontology of the data mining process itself to guide a miner to select approach tools and consider appropriate data elements.

A second, possibly more significant challenge facing the data mining process arises from the so-called "curse of dimensionality." In fact, this is the challenge we attempt to address here. While studying control processes, Richard Bellman (the inventor of dynamic programming) noted that, with the exponential increase in volume of a sample space as the number of attributes (or dimensions) increases, we are faced with an accompanying requirement to have exponentially increasing data to characterize the higher dimensional space (Bellman, 1961). In traditional data mining problems, the data exists (i.e., there are large numbers of data records, even in a high-dimensional space), but the challenge comes in analyzing this huge amount of data to extract interesting and useful knowledge.

Mediated Data Collection and Integration

The diagnostic maturation process requires ready access to design, maintenance, and other logistic support information sources. Data essential to analysis of maturation issues may be generated by system producers (e.g., engineering, product support organizations, etc), or by system users (maintenance and supply chain data, etc). Each of these organizations is also multifaceted in nature. Engineering organizations, for example, are composed of sub-disciplines such as design, reliability and maintainability, etc. Compounding the problem is the fact that the data of interest resides in multiple systems each with different owners where it does exist – and it should be recognized that some data that is desirable to have might not be captured in data systems at all. The heterogeneous nature of these sources present issues having to do with access, accuracy, semantic understanding, completeness, and correlation of relevant design features with performance data, and spans hardware platforms, Data Base Management Systems (DBMS), and software standards. More specific data structural concerns may include general data representation issues (e.g., methodologies, hierarchical dominance across systems, uniqueness identification, data types) and more specific data format disparities (e.g., units of measure, code sets, "intelligent" values), and semantic differences in the relationship between data labels (terms) and their intended meaning (concepts). This heterogeneity occurs quite naturally as multiple systems are developed and evolved as a function of independent decisions and design within the development lifecycle of a system. The physical constraints having to do with access are easing, but generally accepted approaches to content integration are only recently appearing in actual applications.

Federated data servers, or query engines, can address many of these heterogeneity issues. The goal of a federated

data server is to provide real-time, integrated access to diverse and distributed data as if it were a single source, regardless of format, location, or operating environment (Hayes and Mattos, 2003). Federated servers address differences in DBMS, physical data structure, structured vs. semi- or unstructured data, data type casting, and query performance across systems. These sort of mechanical or structural issues are essential to integration of the information required for maturation analysis, but they do not address the conceptual issues involving semantic heterogeneity.

Mediated approaches to semantic integration can help by enumerating archetypes of the concepts of interest in the domain of interest – in our case health management and related maintenance and logistic information – then detailing the relationships and constraints between these concepts in a unified ontology – thereby reducing information requests to operations on a single model of the requisite information and mapping those requests between the ontology and logical models of the data sources which can instantiate the information requests.

An inherent advantage of a mediated implementation is in the knowledge that can be synthesized from the models and mappings. An ontology is developed to explicitly represent the semantics of the target information domain, then mappings are created to correlate source data with the terminology and concept relations in the ontology, providing a basic semantic interpretation of the data being accessed and its relation and relevancy to the domain of interest.

The domain ontology subsumes the semantics of the data across multiple data systems but is otherwise independent of the system data representations. From end users' perspectives, a mediated information integration system looks like a single information resource, containing all of the available information within a specific domain.

Ontology-Directed Diagnostic Maturation

We proceed from the assumption that the diagnostic maturation process is fundamentally a data mining/knowledge discovery process. Specifically, we claim that there are three major types of maturation steps to be performed relative to system diagnosis, all with the goals of either improving the accuracy of diagnosis or improving the efficiency of the diagnostic process.

Refining cost estimates of actions or tests being performed in the maintenance process. These cost estimates are used to optimize the maintenance process and, if inaccurate, can substantially weaken the benefits expected from a strong optimization algorithm.

Refining the probabilities of occurrence of various maintenance events. Again, these probabilities are essential to the optimization process since we are attempting to minimize expected cost over the system being maintained. Inaccurate probabilities can skew the process in ways that can yield significant, sub-optimal maintenance procedures.

Correcting errors in an underlying system model. In fact, many types of errors can exist in these types of models; however, most diagnostic models tend to focus on capturing relationships between observable information (i.e., tests) and the diagnoses to be drawn (i.e., faults). If we restrict ourselves to these types of errors, then our concern becomes determining if there are relationships that are either missing from the model or if relationships need to be added to the model that currently do not exist. A variation of these two extremes is the case where relationships have qualifying information, and that qualifying information is not completely accurate.

Historically, refining cost or probability estimates has been relatively straightforward, so we focus our discussion in this paper on improving the accuracy of the system diagnostic models.

Ontologies for Diagnostic Maturation

For the following discussion, we draw on the development of a standard ontology for diagnostic maturation being developed by the Institute of Electrical and Electronics Engineers (IEEE), recognizing that alternative ontologies are possible. The approach we describe in the following should work, in principle, with any of these alternative ontologies.

Currently, the IEEE Standards Coordinating Committee 20 (SCC20) of the IEEE Standards Association is developing two families of standards that together define an ontology of the diagnostics problem domain. IEEE 1232 Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) defines an ontology specifically covering the diagnostic process itself (IEEE, 2007a). Given the fact the intent is to improve diagnostics, we need the diagnostic ontology as a framework; however, we also need an ontology for the maintenance data collection process so we can mediate these processes to mature the diagnostics. To support this, IEEE P1636 Software Interface for Maintenance Information Collection and Analysis (SIMICA) is being developed (IEEE, 2007b). Currently, SIMICA consists of two "component" standards—IEEE P1636.1 Test Results and IEEE P1636.2 Maintenance Action Information. The Test Results standard provides ontological information about the test process and provides a framework for capturing specific measurements and outcomes of actual tests (IEEE, 2006). The Maintenance Action Information standard provides ontological information about the maintenance process, given supporting components for test and diagnosis (IEEE 2007c). The combination of the three—test, diagnosis, and maintenance—provide an integrated, mediated view of information necessary for process improvement and maturation.

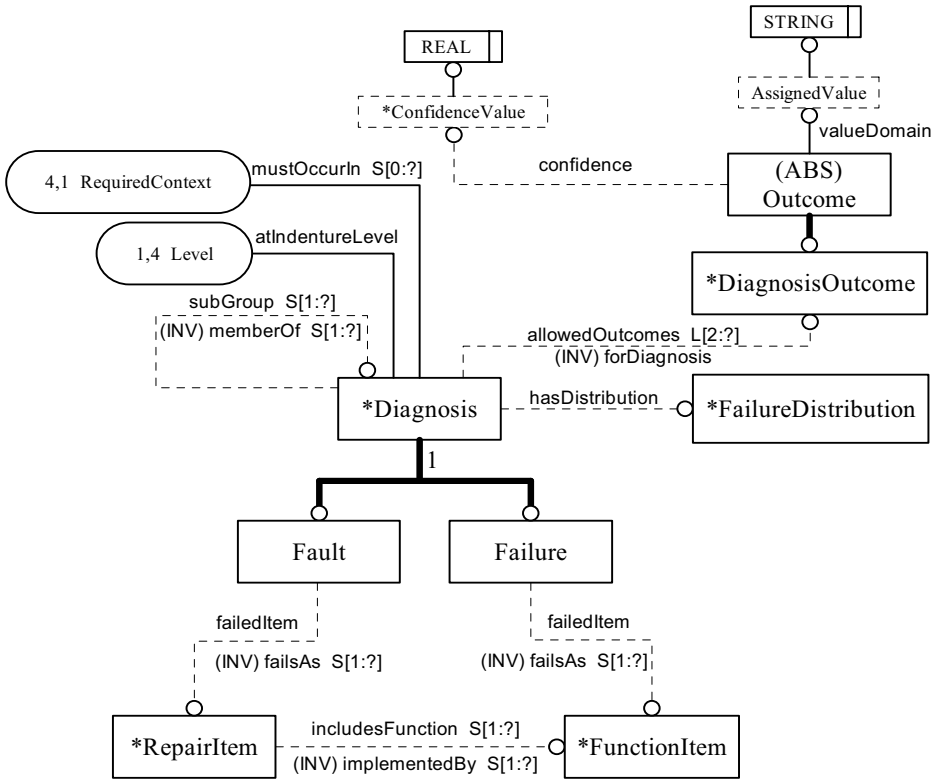


Figure 2. AI-ESTATE Ontology Excerpt: Diagnosis

As an example of how the ontology can guide the data mining process, we will consider the problem of correcting the relationships in a diagnostic model. Currently, AI-ESTATE defines three different types of diagnostic models—the fault tree, the diagnostic inference model, and the Bayes model. For our purposes, we will focus on the Bayes model. Second, AI-ESTATE includes a Dynamic Context Model (DCM), which was designed to provide an information interface to a diagnostic reasoner and represent historical information captured during an actual diagnostic process. All of the entities defined in the DCM are tied back to the third type of model in AI-ESTATE. This model—the Common Element Model (CEM)—provides a top-level ontology for diagnostic information to characterize relationships and constraints between different elements in the diagnostic domain. In fact, the specific diagnostic models are also tied back to the CEM.

Figure 2 shows a small portion of the AI-ESTATE CEM ontology corresponding to a diagnosis. Of interest here is the fact that diagnosis is defined to have a lattice organization where a particular diagnosis can have multiple children and multiple parents. The parent-child relationship is defined here to represent a grouping of diagnoses into, for example, replaceable unit groups or common functional groups. The diagnosis entity also has two attributes of interest to us—mustOccurIn (which is of type RepairContext) and atIndentureLevel (which is of type Level). A similar structure appears for the Test, Action, and Repair entities. In fact, Test and Repair are represented as subtypes of Action.

Remember that our concern is managing the curse of dimensionality. In this case, we can restrict our analysis to a set of diagnoses that are at the same level of indenture, part of the same repair context, and at the same depth within the lattice structure by constructing ontological queries against the mediated information, thus restricting the information to be limited to the subset of interest. Alternatively, we may find that there is a group of diagnoses at the same level of indenture and in the same repair context but appearing at different depths in the lattice structure. We may decide to restrict search to the subset of those elements having a common ancestor in the structure. One final example arises by noting that there are additional entities within the ontology that have attributes of type Level and RepairContext. These common attributes can be used to consider relationships among entities other than tests or diagnoses but appearing within a common context.

In the following, we will assume we have received a data set to be mined corresponding to data collected using

```
function Apriori(D, s)
  L_1 ← set of large 1-itemsets
  k ← 2
    while L_{k−1} not empty do
      C_k ← select(L_{k−1}) // candidate itemset
      for all d in D do
        C_d ← subset(C_k, d) // candidate contained in data
        for all c in C_t do
          count[c] ← count[c] + 1
      L_k ← {c ∈ C_k | count[c] ≥ s} // test for minimal support
      k ← k + 1
  return ∪_k L_k
```

Figure 3. *Apriori* Algorithm for Finding Large Itemsets (Agrawal and Srikant, 1994)

P1636.1 Test Results coupled with the P1232 Dynamic Context Model (for diagnostic history) and P1636.2 Maintenance Action Information (for actual repair information). Using our ontology, we restrict analysis to the set of test results, diagnoses, and repairs occurring at a particular level of indenture with a common repair context that have been verified through the repair certification process.

Guided Association Rule Mining

Given the now restricted set of data to consider, we start by deriving association rules from the data (Agrawal and Srikant, 1994). An association rule is a rule written as in implication of the form $X \Rightarrow Y$ where *X* is the conjunction of a set of variables and *Y* is the conjunction of a different set of variables. Given an association rule and a training data set, we say that the *support s* of the rule is the percentage of examples in the training set for which the conjunction $X \wedge Y$ holds, and the *confidence c* of the rule is the percentage of those examples for which *X* holds where $X \wedge Y$ also holds (i.e., $|X \wedge Y| / |X|$). Note that support can be interpreted as "coverage" (i.e., the number of examples that match all of the variables in the rule) and confidence can be interpreted as "accuracy" (i.e., the percentage of examples for which the $X \Rightarrow Y$ holds).

Agrawal and Srikant provide an algorithm, called *Apriori*, with an efficient variant, called *AprioriHybrid* for finding association rules in a data set. Both algorithms require that minimum support and confidence thresholds be set by the user. The algorithm finds so-called "large itemsets" which are sets of variables for which the minimum support has been obtained. We refer to a large itemset with *k* variables to be a *k*-itemset. A basic version of the *Apriori* algorithm can be found in Figure 3. In this algorithm, let L_k be the set of large *k*-itemsets, C_k be the set of candidate *k*-itemsets, *d* is some subset of "literals" or "items" in the data (called transactions in the original), and *s* be the user-specified minimal percent of transactions containing some candidate itemset *c*. This algorithm then finds the itemsets of various sizes meeting the minimum support threshold. Once the itemsets are found, it is a simple matter to construct the rules where the left hand sides are itemsets that are subsets of itemsets covering the entire rule.

The *AprioriHybrid* algorithm implements the *Apriori* algorithm more efficiently. Specifically, by using breadth-first search and a hash tree, the itemsets exceeding the minimum support threshold (and subsequent association rules satisfying the minimum confidence threshold) can be found in time linear in the number of examples. Given the *AprioriHybrid* algorithm, diagnostic maturation proceeds by applying the algorithm to the data set that has been restricted according to our ontology. Association rules can be constructed relating tests to one another, tests and diagnoses, or even tests/diagnoses with other factors in the maintenance process (such as a repair action). While the general *AprioriHybrid* algorithm permits construction of rules of the form $P_1 \wedge \ldots P_n \Rightarrow Q_1 \wedge \ldots \wedge Q_m$, we will restrict the rules of interest to use to those containing only one consequent. These association rules can then be ranked by their confidence and used to refine the diagnostic model as long as the addition of such rules improve the accuracy of diagnosis.

Augmenting Bayesian Classifiers for Diagnosis

Consider a specific case where we have a Bayesian diagnostic model as defined in (IEEE, 2007a). Previous work by Sheppard *et al.* has demonstrated that naïve Bayesian classifiers can provide a relatively simple method for capturing diagnostic relationships that also yields relatively accurate diagnostics (Sheppard, Butcher, Kaufman and MacDougall, 2006). However, naïve Bayes models are only capable of representing linearly separable concepts, and even then, only a subset of all linearly separable concepts can be represented (Zhang, Ling, and Zhao, 2004). It is also interesting to note that the popular *D*-matrix of model-based diagnosis suffers from the same problem (Sheppard and Butcher, 2007).

Formally, the naïve Bayes network finds the class label (i.e., diagnosis) that maximizes the *a posteriori* probability of the specific class given the set of observations (i.e. tests):

$$D = \arg\max_{d_i \in \mathbf{D}} P(d_i \mid o(T_1),\ldots,o(T_n)) = \arg\max_{d_i \in \mathbf{D}} P(o(T_1),\ldots,o(T_n) \mid d_i)P(d_i).$$

where d_i is the ith diagnosis, T_j is the jth test, and $o(T_j)$ is the observed outcome for T_j. But we have a problem that the joint distribution over the tests is exponential in the number of tests. The naïve Bayes assumption states that we can treat each of these observations as if they are conditionally independent given the diagnosis, and this leads to the classification rule:

$$D = \arg\max_{d_i \in \mathbf{D}} P(d_i)\prod_{j=1}^{n} P(o(T_j) \mid d_i).$$

Given a set of training data mapping test results (outcomes) to actual faults repaired, we can "learn" the naïve Bayes network by observing that $P(d_i)$ is simply the frequency of occurrence of a particular fault in the data set and similarly, $P(o(T_j) \mid d_i)$ is the frequency of test outcome $o(T_j)$ considering only the particular diagnosis d_i.

We proceed under the assumption the initial diagnostic model that has been deployed is a naïve Bayes network. This is reasonable for an initial deployment given empirical evidence of frequent effectiveness of such a model; however, the theoretical limitations indicate that the model can be improved as data is collected. One of the primary methods for improving naïve Bayes networks is by "augmenting" the networks with additional conditional dependence relationship that had previously been assumed away.

One popular approach augmenting naïve Bayes networks is through the tree-augmented naïve Bayes (TAN) algorithm (Friedman, Geiger, and Goldszmidt, 1997) Empirical evidence has shown that the improvements from TAN can be substantial; however, the experiments in (Sheppard, J., Butcher, S., Kaufman, M., and MacDougall C. 2006) suggest that such improvements may, on average, be marginal. We hypothesize that one reason for such marginal improvement is the fact that the augmentation algorithm can still miss important dependencies while including lesser dependencies that can actually deceive the classifier. This hypothesis is being explored in separate work.

Proceeding from the above hypothesis, we further hypothesize that using association rules can provide a better indication of needed augmentations for the network. The task then becomes, for a particular association rule, how do we incorporate that rule into the current diagnostic model? Specifically, for an association rule of the form $P_1 \wedge \ldots P_n \Rightarrow Q$, we include (or update) the probability table corresponding to $P(Q \mid P_1,\ldots,P_n)$. There are three cases to consider.

Case 1—A new association rule relating tests and diagnoses: This case is the simplest of the three since it simply revises an existing conditional probability table in the current network. Specifically, if we have a rule $D \Rightarrow T$, we modify the conditional probabilities for $P(o(T) \mid D)$ according to the data in the training set. Note that, if we have the opposite rule of $T \Rightarrow D$, then we can consider the contrapositive and model $\neg D \Rightarrow \neg T$, which still leads to a simple update of the current conditional probability table.

Suppose we have a more complex rule of the type $D_i \wedge D_j \Rightarrow T$. This, in fact, corresponds to a multiple fault, which can now be added to the model by including the conditional probability table for $P(o(T) \mid D_i, D_j)$. This constitutes an augmentation to the network.

Case 2—A new association rule relating multiple tests: In this case, any rule identified, whether $T_i \Rightarrow T_j$ or the more general $T_1 \wedge \ldots \wedge T_m \Rightarrow T_j$ results in an augmentation of the network because we now must add a new conditional probability table to the network. Note that this can be done directly by following the template of $P(T_j \mid T_1,\ldots,T_m)$. Note, however, that the actual probability tables would correspond to $P(T_j \mid T_1,\ldots,T_m,D_k)$ for all diagnoses D_k because the information needs to be merged with the existing model.

Case 3—A new association rule relating a test or diagnosis to some other variable: Finally, this case is interesting because it not only augments the network with an additional conditional probability table, but it incorporates an additional random variable into the network. The new random variable corresponds to the factor not currently captured by the set of tests or diagnoses, such as an in-process repair action. The first step, then, is to add the new variable to the network and then add the associated probability table.

In considering the above approach, there are several computational issues to be considered. First, as augmentations are included in the network, the size of the corresponding conditional probability tables grows exponentially. This is one of the reasons for selecting the minimum support and confidence values carefully. Second, even controlling the number of augmentations, the computational expense of processing augmenting networks also grows exponentially. Therefore, utilizing resulting networks may require alternative processing approaches such as conversion to join trees, Monte Carlo simulation, or some alternative form of network transformation. Third, collecting further historical information may determine that a previously learned association rule does not, in reality, apply. At such time, the rule and associated portions of the probability tables affected should be removed.

Future Directions

The presented algorithm is an initial approach to maturing diagnostic models based on an associated ontology for the

test, diagnostic, and maintenance process. This approach is work in progress, and the first step is to run several experiments to validate the approach. It is reasonable to expect the model to perform relatively well given that it proceeds from an existing model and only modifies the model based on statistical information collected in the field. The key advantage to the approach is that it provides a directed strategy for using the ontology to guide the maturation process. Additional future work would include extending the idea to other parts of the maintenance process and other parts of the ontology.

In addition to augmented Bayes learning, there are many machine learning methods available for creating and revising classification models. These methods apply directly to the problem of maturing diagnostic models, which are also classification models. Therefore, a rich area of related research is to use the ontology to direct learning and revision of models such as decision trees (P1232 fault tree model), rule sets (P1232 diagnostic logic model) or companion models such as neural networks, support vector machines, or even hidden Markov models (for system prognosis).

Another interesting question is whether the approach can be used from an alternative perspective where the ontology directs search in areas where no relationships in the ontology itself exist. The purpose here would be to determine if there are correlations from the data indicating a previously unknown relationship, thus permitted maturation of the ontology itself. Specifically, the KD process might identify a useful relationship that was not explicitly modeled in the original ontology.

Finally, one of the authors is exploring the utility of constructing separate system-level diagnostic models corresponding to specific or subsets of a fleet of systems given empirical evidence to show that such models can yield more accurate diagnosis than a single model covering all instances of a given system. Based on contextual information (which is included in the ontologies), work is ongoing to utilize this contextual information to better determine what historical data to apply to mature or develop a given model.

Conclusion

The purpose of this paper was to present a framework for utilizing ontologies combined with machine learning to mature diagnostic models. The approach focuses on using the ontologies to restrict data sets in a meaningful way to manage the curse of dimensionality and still yield useful model revisions. While the research is still too early to report experimental results, theoretical analysis suggests the approach has promise.

References

Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. *Proceedings of the 20th VLDB Conference*, Santiago, Chile.

Bellman, R. 1961. *Adaptive Control Processes: A Guided Tour*, Princeton, NJ: Princeton University Press.

Bernstein, A., Hill, S., and Provost, F. 2002. "Intelligent Assistance for the Data Mining Process: An Ontology-Based Approach," CeDAR Working Paper IS-02-02, Center for Digital Economy Research, Stern School of Business, New York University,.

Friedman, N., Geiger, D., and Goldszmidt, M. 1997. Bayesian Network Classifiers. *Machine Learning*, 29:131–163.

Han, J. and Kamber, M. 2006. *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers.

Hayes, H. and Mattos N. 2003. Information On Demand. *IBM DB2 Magazine*, Quarter 3, Vol. 8, Issue 3.

IEEE 2007a. P1232 *IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*, Draft 1.0, Piscataway, NJ: IEEE Standards Press.

IEEE 2007b. P1636 *IEEE Standard Software Interface for Maintenance Information Collection and Analysis (SIMICA)*, Draft 1.0, Piscataway, NJ: IEEE Standards Press.

IEEE 2006. P1636.1, *IEEE Trial Use Standard Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Test Results and Session Information via the Extensible Markup Language (XML)*, Draft 1.0, Piscataway, NJ: IEEE Standards Press.

IEEE 2007c. P1636.2, *IEEE Trial Use Standard Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Maintenance Action Information via the Extensible Markup Language (XML)*, Draft 1.0, Piscataway, NJ: IEEE Standards Press, 2007.

Quinlan, J. R. 1990. "Learning Logical Definitions from Relations," *Machine Learning*, 5:239–266.

Sheppard, J., Butcher, S., Kaufman, M., and MacDougall C. 2006. Not-So-Naïve Bayesian Networks and Unique Identification in Developing Advanced Diagnostics. *Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana, March.

Sheppard, J. and Butcher, S. 2007. A Formal Analysis of Fault Diagnosis with D-Matrices, to appear in *Journal of Electronic Testing: Theory and Applications*, Springer.

Wilmering, T. 2001. Semantic Requirements on Information Integration for Diagnostic Maturation. *IEEE AUTOTESTCON Conference Record.*

Wilmering, T. 2003. When Good Diagnostics Go Bad – Why Maturation is Still Hard. *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT.

Zhang, H., Ling C., and Zhao Z. 2004. The Learnability of Naïve Bayes. *Advances in Artificial Intelligence: 13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Quebec, Canada, Springer, LCNS Vol 1822, pp. 432–441.

Posters

State Tracking in the Mode Space

Mehdi Bayoudh and **Louise Travé-Massuyès**
Université de Toulouse
LAAS-CNRS, FRANCE
{bayoudh, louise}@laas.fr

Xavier Olive
THALES ALENIA SPACE, FRANCE
Xavier.Olive@thalesaleniaspace.com

Abstract

The behavior of complex systems is generally a mix of continuous and discrete dynamics that calls for new concepts and methods. In particular the corpus of model-based diagnosis tools does not apply directly but requires some extension to be applied to such systems. This paper considers hybrid systems represented by hybrid automata, in which every state stands for an operational mode of the system and is associated a set of algebraic and differential equations to describe the continuous behavior in that mode. It uses an extension of the parity space approach in the form of a set of residuals associated to each mode and proposes a geometrical interpretation that makes clear the contribution of each local residual vector to the mode identification problem. New concepts of mutual and 3rd-diagnosability are introduced. A method for tracking the state of the hybrid system is then presented in this framework.

Introduction

The use of modern technologies like embedded electronic controllers in physical processes, is increasing, and lead to complex systems to mix both discrete and continuous behavior, with high demands on performance and availability. This paper incorporates the capability of model-based diagnosis into an hybrid estimation scheme using hybrid automata represented in the mode space. The discrete behavior is modeled by controlled and autonomous (observed or not) transitions. The continuous behavior is represented by algebraic and differential equations that describe the continuous evolution in each operating mode and lead to analytic redundancy relations, used to model constraints linking continuous observable variables (whose values are given by sensors).
We propose a geometrical representation based on the mode space which allows us to abstract the continuous behavior in the discrete state space, mapping between modes and space regions. This framework offers a nice geometrical interpretation of the mutual diagnosability property, which is defined as a constituant of diagnosability for multimode continuous systems.
We show that even in the case in which the continuous and the discrete event systems underlying a hybrid system are both non diagnosable (Travé-Massuyès *et al.* 2004) (Cocquempot, Mezyani, & Staroswiecki 2004) (Sampath *et al.* 1995), the hybrid system can be diagnosable. In this paper, this is illustrated by a tracking algorithm that couples Continuous Systems (CS) and Discrete Event Systems (DES) diagnosis techniques and a appropriately chosen diagnosis scenario.
The first section defines the hybrid-modeling framework, the second section presents the geometrical interpretation and the mapping between operational modes and regions of the mode space. Section 3 proposes new concepts of mode signature and introduces multimode systems diagnosability properties. Section 4 proposes a hybrid diagnosis approach aimed at tracking the system in the mode space by taking advantage of both continuous and discrete dynamics. Finally, Section 5 illustrates the approach with an illustrative example.

Hybrid Systems Modeling

As mentioned in (Henzinger 1996) (Hofbaur & Williams 2004), a hybrid system may be described by a hybrid automaton defined as a tuple $S = (\zeta, Q, \Sigma, T, C, (q_0, \zeta_0))$.
Where:

- ζ is the set of continuous variables, which include observable and non observable variables. The set of observable variables is denoted by ζ_{OBS}.
- Q is the set of discrete system states. Each state $q_i \in Q$ represents a functional mode of the system.
- Σ is the set of events. Events, noted e_i, correspond to command value switches, spontaneous mode changes and fault events.
- Σ is partitioned in a set of observable events Σo and a set of unobservable events Σ_{uo}. Fault events Σ_F are unobservable.
- T is the transition function, Q$\times\ \Sigma \rightarrow Q$.
- C is the set of system constraints linking continuous variables. Here, these constraints are expressed in terms of

Analytic Redundancy Relations (ARRs). ARRs are computed from the continuous behavior of each mode (state and observation equations in the continuous state space) and only involve observable variables. We associate a subset of relations $ARR_i \subseteq C$ to a functional mode q_i.

- $(\zeta_0, q_0) \in \zeta \times Q$, is the initial condition.

The Underlying Discrete Event System

The discrete part of the hybrid automaton, given by $M = (Q, \Sigma, T, q_0)$, is a discrete automaton that describes the discrete dynamics of the system, i.e. the possible evolutions between operating modes of Q. Modes include nominal and fault modes as well as an unknown mode, which stands for all the non anticipated faulty situations. The unknown mode has no specified behavior and hence no associated ARRs.

The Underlying Continuous System

The continuous part of the hybrid automaton is given by the continuous models associated to each mode q_i (nominal or anticipated fault modes).

$$\begin{cases} X_i(n+1) &= A_i X_i(n) + B_i U(n) \\ Y(n) &= C_i X_i(n) + D_i U(n) \end{cases}$$

with:

- $X_i(n)$: the state vector at time step nT_s.
- $U(n)$: the input vector at time step nT_s.
- $Y(n)$: the output vector at time step nT_s.

T_s, is the sampling period; A_i, B_i, C_i and D_i are constant matrices of appropriate dimensions. The underlying continuous system $\Xi = (\zeta, Q, C, \zeta_0)$ (also called *the multimode system*) describes the continuous behavior of the system. Consequently, discrete transitions become transparent and only continuous information is available for diagnosis and diagnosability analysis.
Following the parity space approach, consistency tests may take the form of a set of Analytical Redundancy Relations by eliminating non observable variables (Cordier *et al.* 2004). The ARRs set associated to a mode q_i are denoted by ARR_i. An Analytic Redundancy Relation ARR_{ij} can be expressed as $r_{ij} = 0$, where r_{ij} is called the residual of the ARR. Since ARRs are constraints that only contain observable variables, they can be evaluated on-line with the incoming observations given by the sensors, allowing one to check the consistency of the observed against the predicted system's behavior. ARRs are satisfied if the observations satisfy the model constraints, in which case the associated residuals are zero. In the opposite case, all or some of the residuals are non zero. The set of residuals in mode q_i hence results in a local Boolean fault indicator tuple.

$$r_{ij} = \begin{cases} 0 & \text{when } AAR_{ij} \text{ is satisfied} \\ 1 & \text{otherwise} \end{cases}$$

$j = 1, ..., N_{ARR(q_i)}$, where $N_{ARR}(q_i)$ is the number of associated ARRs/residuals.

In the case of linear systems, the Analytic Redundancy Relations can be computed using the *Parity space approach* (Staroswiecki & Comtet-Varga 2001). This approach has been recently extended to multimode systems in (Cocquempot, Mezyani, & Staroswiecki 2004).
Given a vector V, let us denote by V^p the vector obtained by the concatenation of the vector values at every sampling instant $(n-p+i)$, $0 \leq i \leq p$, for a given order p.
Hence $V^p(n) = [V^T(n-p), ..., V^T(n-p+i), ..., V^T(n)]^T$.
Consider a multimode system like Ξ and a mode q_i, then the computational form of the residual vector $R_i = [r_{i1}, r_{i2}, ..., r_{iN_{ARR}(q_i)}]^T$ at order p_i is:

$$\rho_{c_i}^{p_i}(n) = \Omega_i^{p_i} Y^{p_i}(n) - \Omega_i^{p_i} L_i^{p_i}(A_i, B_i, C_i, D_i) U^{p_i}(n)$$

and its evaluation form is :

$$\rho_{e_i}(n) = 0$$

with:

$$L_i^{p_i}(M_i, N_i, P_i, Q_i) = \begin{pmatrix} Q_i & 0 & ... & 0 \\ P_i N_i & Q_i & ... & ... \\ ... & ... & ... & 0 \\ P_i M_i^{(p_i-1)} N_i & ... & P_i N_i & Q_i \end{pmatrix}$$

$$O_i^{p_i} = \begin{pmatrix} C_i \\ C_i A_i \\ ... \\ C_i A_i^{p_i} \end{pmatrix}$$

and $\Omega_i^{p_i}$ is a matrix orthogonal to $O_i^{p_i}$(there always exists an order p_i such that $O_i^{p_i}$ exists).
In the multimode framework, the set of ARRs linked with each functional system mode is generally different, although some ARRs may be shared.
For sake of clarity and simplicity, in the paper we assume that the parity space order is the same for all modes and that it is equal to p.

Geometrical Interpretation in the Mode Space

In this section, we propose a new representation of hybrid systems, in the vectorial space $\mathbb{R}^m$, where m is the number of operational system modes. The vectorial space

$\mathbb{R}^m$ is described by the base $B = (\vec{q_1}, \vec{q_2}, ..., \vec{q_m})$ and it is called *the mode space*, with $\vec{q_1} = [1, 0, 0, ..., 0]_m$, $\vec{q_2} = [0, 1, 0, ..., 0]_m$, ..., $\vec{q_m} = [0, 0, 0, ..., 1]_m$.
It will be shown in the next section that this vectorial representation of the hybrid space provides a framework to geometrically interpret mutual diagnosability, also called discernability in (Cocquempot, Mezyani, & Staroswiecki 2004).
When the system mode is q_i, the associated residual vector is zero ($R_i = [0, 0, ..., 0]^T$). The reciprocal is not true, it depends on the system's mutual diagnosability that is presented in the following section.
At a given time step, the residuals are tested by comparing the computation and the evaluation forms (ρ_{c_i} and ρ_{e_i}) by using a set of observable variables measured during p time steps.
The behavior of the system in the mode space at time step nT_s (T_s is the sampling period) can be described by the linear mapping:

$$||R_1||(n)\vec{q_1} + ||R_2||(n)\vec{q_2} + ... + ||R_m||(n)\vec{q_m}$$

where $||R_i|| = \sqrt{(\sum_{j=1..N_{ARRs(q_i)}} r_{ij}^2)}$.

Mapping between Discrete Modes and Regions of the Mode Space
Since $||R_i|| = 0$ when the system is in mode q_i, at time step nT_s, the system evolves in the space region $\mathcal{R}^{q_i}$, contained in the subspace orthogonal to the vector $\vec{q_i}$, i.e. $\mathcal{R}^{q_i}$ is included in the vectorial subspace generated by the base $(\vec{q_1}, ..., \vec{q_{i-1}}, \vec{q_{i+1}}, ..., \vec{q_m})$. The dimension of $\mathcal{R}^{q_i}$ depends on the mutual diagnosability property of the multimode system detailed in the next section.
Figure 2 presents an example of the mode space in the 3D space for a 3-mode system described by the automaton of figure 1.

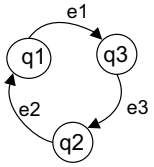


Figure 1: The automaton of a 3-modes hybrid system

Diagnosability of the underlying continuous system

Diagnosability properties known for DES on one hand and for CS on the other hand cannot be applied directly on hybrid systems. In this section, we present a theory introduced to analyze the diagnosability of the underlying continuous system. It relies on the new concepts of *mirror and mode signatures*. These concepts are then used to establish the definition of the diagnosability of multimode systems (Bayoudh & Travé-Massuyès 2007).

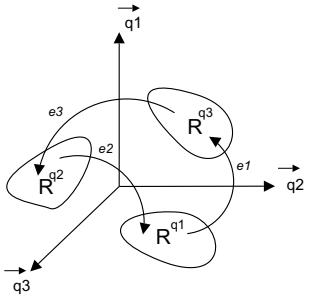


Figure 2: Example of a 3-dimensional mode space

Mirror and Reflexive signatures

In this subsections, the concepts of mirror signature and reflexive signature are defined and used to define mode signatures.

Definition 1 *Mirror Signature*
Given the vector $R_k = [r_{k1}, r_{k2}, ..., r_{kN_{ARR(q_k)}}]^T$ *of system residuals in mode* q_k*, the* q_k*-mirror signature of mode* q_j *is given by vector* $S_{j/k} = [s_{1_{j/k}}, ..., s_{N_{ARR(q_k)_{j/k}}}]^T = [R_k(\zeta_{OBS}^j)]^T$.

The q_k-mirror signature of mode q_j is the vector of residuals of mode q_k computed with the observations ζ_{OBS}^j obtained when the system is in mode q_j. We say that it is the signature of mode q_j seen in (the mirror of) mode q_k.

Definition 2 *Reflexive Signature*
Given the vector $R_j = [r_{j1}, r_{j2}, ..., r_{jN_{ARR(q_j)}}]^T$ *of system residuals in mode* q_j*, the reflexive signature of mode* q_j *is given by vector* $S_{j/j} = [0, 0, ..., 0]^T = [R_j(\zeta_{OBS}^j)]^T$.

The reflexive signature of mode q_j is the vector of residuals of mode q_j computed with the observations ζ_{OBS}^j obtained when the system mode is q_j. Note that the reflexive signature of a mode is actually its own mirror signature.

Mode signatures

Definition 3 *The mode signature of a mode* q_j *is the vector obtained by the concatenation of all its mirror signatures,* $Sig(q_j) = [S_{j/1}^T, S_{j/2}^T, ..., S_{j/j}^T, ..., S_{j/m}^T]^T$.

From mode signatures to the diagnosability of the multimode system definition

The mode signature concept, presented above, leads us to define the diagnosability of multimode systems.

Remark
In our approach, the faulty behaviors (but the unknown mode) are modeled by fault modes, thus an anticipated fault is diagnosable if the associated fault mode is diagnosable.

Definition 4 *Two modes q_i and q_j ($i \neq j$) are diagnosable iff $Sig(q_i) \neq Sig(q_j)$.*
The multimode system Ξ is diagnosable iff all pairs of modes q_i and q_j, $i \neq j$, are diagnosable.

For two modes q_i and q_j, diagnosability can be interpreted along two complementary ways:

- *Mutual Diagnosability*: two modes q_i and q_j, $i \neq j$, are mutually diagnosable iff $S_{i/j} \neq S_{j/j}$ or $S_{j/i} \neq S_{i/i}$.
  The multimode system is mutually diagnosable iff for all pairs of modes q_i and q_j, $i \neq j$, $S_{i/j} \neq S_{j/j}$ or $S_{j/i} \neq S_{i/i}$.
- *3rd-Diagnosability*: q_i and q_j are q_k-3rd-diagnosable iff they have different signatures w.r.t. mode q_k, i.e. different q_k-mirror signatures, $k \neq i, j$. Formally, q_i and q_j, $i \neq j$, are 3rd-diagnosable iff $\exists k \neq i, j$ such as $S_{i/k} \neq S_{j/k}$.
  The multimode system is 3rd-diagnosable iff for all pairs of modes q_i and q_j, $i \neq j$, there exist $k_{i,j}$ such that $S_{i/k_{i,j}} \neq S_{j/k_{i,j}}$.

Actually, it is easy to show that, for two modes q_i and q_j, $i \neq j$, diagnosability requires mutual or 3rd-mirror-diagnosability. Consequently, the multimode system is diagnosable iff for all pairs of modes q_i and q_j, $i \neq j$, mutual or 3-rd diagnosability holds.

The Mutual diagnosability property seen in the Mode Space

Proposition 1 *Two modes q_i and q_j, $(i \neq j)$ are non mutually diagnosable (non discernable) iff $\forall\ \zeta^j_{OBS}$:*

$$||R_j(\zeta^i_{OBS})|| = ||R_i(\zeta^j_{OBS})|| = 0$$

In our case, ζ^j_{OBS} =(input U(t),output Y(t)) obtained when the system mode is q_j.

Proof 1 *q_i and q_j are non mutual diagnosable iff*
$S_{i/j} = S_{j/j} = [0, 0, ..., 0]^T_j$ *and* $S_{j/i} = S_{i/i} = [0, 0, ..., 0]^T_i$
$\Leftrightarrow$
$||R_j(\zeta^i_{OBS})|| = ||R_j(\zeta^j_{OBS})|| = 0$ *and* $||R_i(\zeta^j_{OBS})|| = ||R_i(\zeta^i_{OBS})|| = 0 \Leftrightarrow$
$||R_j(\zeta^i_{OBS})|| = ||R_i(\zeta^j_{OBS})|| = 0$ □

In the linear case, (Cocquempot, Mezyani, & Staroswiecki 2004) proved the following result.

Theorem 1 *Two modes q_i and q_j, $i \neq j$, are non discernable iff $Rank(O^p_i) = Rank(O^p_j) = Rank([O^p_i O^p_j (L^p_i - L^p_j)])$.*

The mutual diagnosability property can easily been interpreted geometrically in our mode space framework, making clear what it means in terms of the regions associated to different modes.

Theorem 2 *A mode q_i is mutually diagnosable from every mode $q_j, (j \neq i)$ iff $dim(\mathcal{R}^{q_i}) = m - 1$.*

Proof 2 *(⇒)*
$\forall i, \mathcal{R}^{q_i} \subseteq$ *the subspace defined by the base* $(\vec{q_1}, ..., \vec{q_{i-1}}, \vec{q_{i+1}}, ..., \vec{q_m}) \Rightarrow dim(\mathcal{R}^{q_i}) \leq m - 1$
q_i is mutually diagnosable from q_j, $\forall j \neq i \Rightarrow$
($\forall j \neq i, ||R_i(\zeta^i_{OBS})|| = 0 \Rightarrow \exists \zeta^i_{OBS}$ *such that* $||R_j(\zeta^i_{OBS})|| \neq 0) \Rightarrow$
$\forall i \neq j, \mathcal{R}^{q_i} \cap \vec{q_j} \neq \{0\} \Rightarrow$
$dim(\mathcal{R}^{q_i}) = m - 1$ □.
(⇐)
$dim(\mathcal{R}^{q_i}) = m - 1 \Rightarrow \forall j \neq i, \exists \zeta^i_{OBS}$ *such that* $||R_j(\zeta^i_{OBS})|| \neq 0$, *hence q_i and q_j are mutual diagnosable* □.

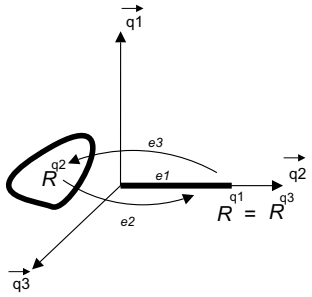


Figure 3: Example of 2 non mutually diagnosable modes (q_1 and q_3) in the 3D mode space

Proposition 2 *If two modes q_i and q_j are non mutually diagnosable then $dim(\mathcal{R}^{q_i}) \leq m-2$ and $dim(\mathcal{R}^{q_j}) \leq m-2$.*

Proof 3 *q_i and q_j are non mutually diagnosable* ⇒
$\forall \zeta^i_{OBS}$ *(=(U(t),Y(t)) when the system mode is q_i),* $||R_i(\zeta^i_{OBS})|| = ||R_j((\zeta^i_{OBS}))|| = 0 \Rightarrow$
$||R_1||\vec{q_1} + ... + ||R_i||\vec{q_i} + ... + ||R_j||\vec{q_j} + ...||R_m||\vec{q_m} =$

$||R_1||\vec{q_1} + ... + ||R_{i-1}||\vec{q_{i-1}} + ||R_{i+1}||\vec{q_{i+1}} + ... + ||R_{j-1}||\vec{q_{j-1}} + ||R_{j+1}||\vec{q_{j+1}} + ...||R_m||\vec{q_m} \Rightarrow$
$dim(\mathcal{R}^{q_i}) \leq m-2$ *and* $dim(\mathcal{R}^{q_j}) \leq m-2$ □

In the example of figure 3, modes q_1 and q_3 are non mutually diagnosable (∀ input U(t) and output Y(t), $||R_1|| = ||R_3|| = 0$). In the hybrid space, non mutual diagnosability shows itself by $\mathcal{R}^{q_1} = \mathcal{R}^{q_3}$, represented by the oriented vector $\vec{q_2}$ ($dim(\mathcal{R}^{q_1}) = dim(\mathcal{R}^{q_3}) = 1$).
Mode q_2, is mutually diagnosable from all other modes, $dim(\mathcal{R}^{q_1}) = 2$.

Coupling the Parity Space Approach and the DES Diagnoser Approach

In this section we propose to enrich the multimode system with the knowledge about the discrete dynamics. In fact, we couple the diagnoser approach –used for DES diagnosis– with the extension of the parity space approach. This allows us to devise a diagnosis algorithm that takes into account both continuous and discrete information. The parity

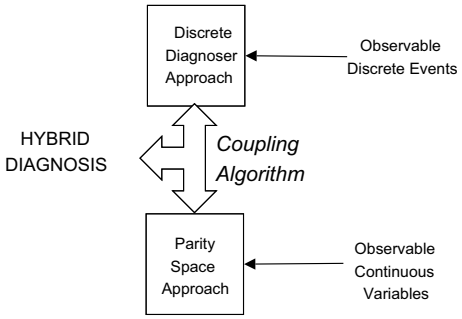


Figure 4: Coupling the Diagnoser and the Parity Space Approach

space approach –used to continuous systems– was presented in section 1, as well as its extension to multimode systems. The DES diagnoser approach is described in the following subsection.

DES Diagnoser Approach

The discrete part of the system is modeled by the finite state machine $M = (Q, \Sigma, T, q_0)$ (see section 1). We consider $\Sigma_F \subseteq \Sigma_{uo}$ as the set of fault events to be diagnosed. We assume that the underlying discrete event system, M, has no unobservable cycles (i.e cycles containing unobservable events only).
The set of fault events Σ_F is partitioned into disjoint sets corresponding to different fault types F_i, $\Sigma_F = \Sigma_{F_1} \cup \Sigma_{F_2} \cup ... \cup \Sigma_{F_n}$ and $\Sigma_{F_i} \cap \Sigma_{F_j} = \emptyset$, $for\, i \neq j$. The aim of the diagnosis is to make inferences about past occurrences of fault types on the basis of the observed events. In order to solve this problem the system model is directly converted into a diagnoser.

The diagnoser $Diag(M) = (Q_{Diag}, \Sigma_{Diag}, T_{Diag}, q_{0\,Diag})$ is a deterministic finite state machine built from the system model $M = (Q, \Sigma, T, q_0)$ (Sampath *et al.* 1995), where:

- $\Sigma_{Diag} = \Sigma_o$ is the set of observable events of the system.
- Q_{Diag} is the set of states of the diagnoser: $Q_{Diag} \subseteq 2^{Q \times 2^{\Sigma_F}}$ or $Q_{Diag} \subseteq \mathcal{P}(Q \times \mathcal{P}(\Sigma_F))$, where $\mathcal{P}(E)$ denotes the power set of E. The states of the diagnoser provide the set of diagnosis candidates as a set of couples whose first element refers to the state of the original system and the second is a label providing the set of faults on the path leading to this state.
- T_{Diag} is the diagnoser transition function built by a recursive process that consists in computing all the reachable states from the diagnoser initial state and by propagating the diagnosis information. For more details see (Sampath *et al.* 1995).
- $q_{0Diag} = \{(q_0, \{\emptyset\})\} \in Q_{Diag}$, is the initial state of the diagnoser.

Definition 5 *Uncertain Diagnoser State.*
Given a diagnoser state $q_{Diag} \in Q_{Diag}$, this state is F_i-uncertain iff F_i does not belong to all the labels of the state whereas F_i belongs to at least one label of the state.

We remind the definition of diagnosability of DES needed for next section.

Definition 6 *DES Diagnosability.*
A fault F is diagnosable iff its occurrence is always followed by a finite observable sequence of events that allows one to diagnose F with certainty (Pencolé 2004). The system is said to be diagnosable iff all the anticipated faults are diagnosable.

Formally, let $s_F t$ be a sequence of events (or trajectory) such that s_F ends with the occurrence of F, and t is a continuation of s_F. F is diagnosable iff:
∀ trajectory $s_F t$, ∃ an integer n: length(t) $\geq n \Rightarrow$ (∀ trajectory s such that $P_{\Sigma_o}(s)$=$P_{\Sigma_o}(s_F t)$, F occurs in s) (Pencolé 2004), where P_{Σ_o} is the projection operator on the set of observable events.

The criterion to check DES diagnosability using the diagnoser is the following:

Theorem 3 *The system M is not diagnosable iff the associated diagnoser $Diag(M)$ contains an uncertain cycle, i.e. a cycle in which there is at least one F_i-uncertain state for some F_i and whose states also define a cycle in the original system M (Sampath* et al. *1995).*

In the DES community the diagnoser is used to analyze the diagnosability property of the system and as an discrete observer.
The state tracking algorithm that we present next uses the diagnoser only as a DES observer, in order to guide the on-line residual computation. Note that in our approach, the diagnosis problem is formulated as a mode estimation across time (state tracking) problem because modes indicate the corresponding fault, if any.

is described in the following subsection.

An Algorithm for State Tracking in The Mode Space

For the hybrid state tracking, we propose an algorithm that allows one to determine the current mode of the system at a given time step. The current mode may be nominal or faulty. The estimation is updated across time so that the algorithm performs both state tracking and diagnosis.
A consistency test (CS-test) based on residual calculation is performed at each cycle of the algorithm. The discrete dynamics (observable or not) are used to guide the on-line residual calculation and reduce the computation time. The algorithm hence couples DES and CS techniques.

Continuous Consistency test checking Consistency test checking consists on the calculation of the computation form of the residuals using observations given by the sensors. Observations are recorded during a temporal window defined by the order of the parity space. The computation form is then compared to the evaluation form (which may include a model of the noise).
A residual is set to 0 when computation and evaluation forms are equal, otherwise to 1.
In this algorithm, only the mutual diagnosability property is used to perform the continuous test. Thus, the algorithm do not require preliminary diagnosability analysis to compute mode signature and can be performed on-line to track the system state.

Algorithm Let us define the following notations:

- $Post(q_i) = \{q \in Q | \exists t \in T \text{ and } e \in \Sigma | t(q_i, e) = q\}$
- $Post_{uo}(q_i) = \{q \in Q | \exists t \in T \text{ and } e \in \Sigma_{uo} | t(q_i, e) = q\}$
- $Q_{Possible} \subseteq Q$ defines the set of possible states of the system during the research phase.

1. **INITIALISATION**
   Current Mode = Initial Mode
2. **OBSERVATION AND FAULT DETECTION LOOP**
   Computation of the Diagnoser state
   Computation of $||R_{CurrentMode}||$
3. **TEST**
   DES-test: Change of the Diagnoser state due to an observable event occurrence ?
   CS-test: $||R_{CurrentMode}|| \neq 0$?
   IF (DES-test = FALSE)

   IF (CS-test=FALSE) GO TO 2
   ELSE $Q_{Possible} = \{q | q \in Post_{uo}(q_{current})\}$

   ELSE $Q_{possible} = \{q | \exists l \in 2^{\Sigma_F} | (q, l) \in q_{diag}\}$
4. $\forall q \in Q_{possible}$, compute $||R_q||$
   $Q_{possible} = Q_{possible} \cap \{q | \ ||R_q|| = 0\}$
   IF ($Q_{Possible} = \emptyset$) then Current Mode = Unknown Mode
   ELSE

   IF (Card($Q_{Possible}$) $=$ 1) then Current Mode = q $\in Q_{Possible}$; GO TO 2

   ELSE [1], FOR every $q \in Q_{Possible}$ DO $\{q_{current} = q$, GO TO 2 $\}$

The algorithm presented above gives the basic principles of the state tracking approach. However, the actual algorithm includes specific procedures to deal with the cases when the system falls into the unknown mode and when several possible trajectories.

Example

We consider a dynamic hybrid system, the underlying discrete system is described by the automaton of figure 5. $o1$ and $o2$ are the observable events (commands, discrete sensor-outputs, ...), $uo1$, $uo2$ and $uo3$ are unobservable events, that can be faulty or not. Modes q_1, q_2, q_3 and q_4

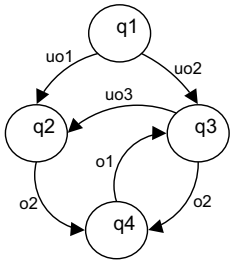


Figure 5: The Underlying Discrete System

can be nominal or anticipated fault modes [2].
The underlying continuous system is given in state space

[1] All the states of $Q_{Possible}$ are non mutually diagnosable, and even the knowledge of the underlying DES dynamics does not disambiguate them. Thus, the algorithm tracks all possible trajectories.

[2] For sake of simplicity, the unknown mode does not appear in the automaton figure but it is implicitly taken into account by the state tracking algorithm.

format, by classical linear dynamic and observation equations. Without loss of generality, we consider no noise and no disturbances (thus the residuals evaluation form is null $\rho_e = 0$).
In the mode $q_i, i \in [1,4]$, the system model in the state space is described by the multimode system:

$$\begin{cases} X_i(n+1) &= A_i X_i(n) + B_i U(n) \\ Y(n) &= C_i X_i(n) + D_i U(n) \end{cases}$$

where

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} -1 & 4 & 0 \\ 0 & -1,5 & 0 \\ 6 & 0 & -2 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -2 & 0 \\ 1 & 0 & -3 \end{pmatrix}, A_4 = \begin{pmatrix} -2 & 1 & 0 \\ -1 & -2 & 0 \\ 2 & 0 & -3 \end{pmatrix}$$

$$B_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix}, B_2 = B_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, B_4 = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}$$

$$C_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, C_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$C_3 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, C_4 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$D_1 = D_2 = D_3 = D_4 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Underlying CS behavior – ARRs Computation For every mode, the order of the parity space is 1, i.e. computational form is calculated from continuous observable variables at time $n-1$ and n, and given as follows:

- $\rho^1_{c_1}(n) = \begin{pmatrix} -0.70 & 0.10 & 0.70 & -0.10 \\ -0.10 & -0.70 & 0.10 & 0.70 \end{pmatrix} \begin{pmatrix} y_1(n-1) \\ y_2(n-1) \\ y_1(n) \\ y_2(n) \end{pmatrix} + \begin{pmatrix} 0.10 & -0.70 \\ -0.70 & -0.10 \end{pmatrix} \begin{pmatrix} u_1(n-1) \\ u_1(n) \end{pmatrix}$

- $\rho^1_{c_2}(n) = \begin{pmatrix} 0.32 & -0.84 & 0.32 & 0.30 \\ 0.20 & 0.44 & 0.20 & 0.84 \end{pmatrix} \begin{pmatrix} y_1(n-1) \\ y_2(n-1) \\ y_1(n) \\ y_2(n) \end{pmatrix} + \begin{pmatrix} -0.93 & -0.32 \\ -1.25 & -0.20 \end{pmatrix} \begin{pmatrix} u_1(n-1) \\ u_1(n) \end{pmatrix}$

- $\rho^1_{c_3}(n) = (-0.34 \quad 0.85 \quad -0.17 \quad 0.34) \begin{pmatrix} y_1(n-1) \\ y_2(n-1) \\ y_1(n) \\ y_2(n) \end{pmatrix} + (0.00 \quad 0.17) \begin{pmatrix} u_1(n-1) \\ u_1(n) \end{pmatrix}$

- $\rho^1_{c_4}(n) = (-0.34 \quad 0.85 \quad -0.17 \quad 0.34) \begin{pmatrix} y_1(n-1) \\ y_2(n-1) \\ y_1(n) \\ y_2(n) \end{pmatrix} + (0.00 \quad 0.17) \begin{pmatrix} u_1(n-1) \\ u_1(n) \end{pmatrix}$

We notice that when the system mode is q_i (or q_j), $||R_3|| = ||R_4|| \ \forall t = nT_s$ (we can verify that $Rank(O^p_3) = Rank(O^p_4) = Rank([O^p_3 O^p_4 (L^p_3 - L^p_4)])$ (theorem 1), and consequently, $\mathcal{R}^{q_3} = \mathcal{R}^{q_4}$. Thus $dim(\mathcal{R}^{q_3}) = dim(\mathcal{R}^{q_4}) = 2 \leq 4-2$ and the modes q_3 and q_4 are non mutually diagnosable.
$dim(\mathcal{R}^{q_1}) = dim(\mathcal{R}^{q_2}) = 4-1$. Thus the modes q_1 and q_2 are mutually diagnosable from all other modes [3] .
The underlying Multimode (Continuous) system is not diagnosable because of the non mutual diagnosability of q_3 and q_4 (We of course noticed that q_3 and q_4 are non 3rd-diagnosable).

DES Behavior – System Diagnoser The diagnoser of the underlying discrete system is given in figure 6. The underlying discrete system is non diagnosable because of the presence of uncertain cycle $(o1, o2)$, and whose states also define a cycle in the original automaton (theorem 3). In

Figure 6: The diagnoser of the underlying discrete system

conclusion, both DES and CS underlying systems are non diagnosable.
Now, by combining DES and CS diagnosis techniques, the algorithm presented in the previous section allows us to track the system state in the mode space and to diagnose the

[3] The mutual diagnosability of modes q_1 and q_2 can also be proven from theorem 1.

occurrence of all unobservable events ($uo1$, $uo2$ and $uo3$). *Example of Scenario: Let us consider that the system follows the trajectory given by (uo1, o2, o1, uo3).*

Results:

$/ * /$
$Intial\ State = q_1$
$Current\ State = q_1$
$Current\ State = q_2\ (\Rightarrow occurrence\ of\ uo_1)$
$Current\ State = q_4\ (\Rightarrow occurrence\ of\ o2)$
$Current\ State = q_3\ (\Rightarrow occurrence\ of\ o1)$
$Current\ State = q_2\ (\Rightarrow occurrence\ of\ uo3)$
$/ * /$

As an example, the detection of the occurrence of the non observable event $uo1$, is established as follows:
1. $CurrentState = InitialState = q_1$
2. Diagnoser state = q_1
$||R_{q_1}|| \neq 0$
3. DES-test = FALSE and CS-test = TRUE
$Q_{possible} = Post_{uo}(q_1) = \{q_2, q_3\}$
4. $||R_{q_2}|| = 0, ||R_{q_3}|| \neq 0$
$Q_{possible} = \{q_2, q_3\} \cap \{q_2\} = q_2$
$CurrentState = q_2$ GO TO 2

This approach hence permits to diagnose the two non mutually diagnosable modes q_3 and q_4 using the discrete dynamics of the system (observable events $o1$ and $o2$). In addition, it permits to diagnose modes q_2 and q_3 using the continuous dynamics of the system ($uo1$ and $uo2$ are non observable, thus q_2 and q_3 cannot be diagnosed using only the discrete dynamics).

Conclusion

In this paper, the proposed representation of the system in the mode space allows one to illustrates state tracking in a vectorial space and provides a geometrical interpretation of the non mutual diagnosability property. Coupling CS and DES diagnosis techniques makes possible to account for both event-based and continuous dynamics in the same hybrid state-tracking algorithm. On one hand, the diagnoser approach guides the on-line residual calculation, and reduces the time of computation by calculating only residuals of modes permitted by the discrete dynamic. On the other hand, the residuals offer an effective consistency test based on the continuous dynamics of the system, and permit to track discrete transitions labeled by non-observable events. In this work, we use the parity space approach to generate residuals but other approaches are possible, more appropriate to non linear systems for example.
Diagnosability analysis of hybrid systems has been analyzed in (Bayoudh, Travé-Massuyès, & Olive 2007) and it has been proved that, taken separately, the diagnosability of the underlying continuous system or of the underlying discrete event system is a sufficient but not necessary condition for the diagnosability of the hybrid system (Bayoudh, Travé-Massuyès, & Olive 2007). The presented example illustrates a case for which both underlying CS and DES are non diagnosable but the hybrid system is. The hybrid tracking algorithm is able to discriminate all the modes because of the coupling of DES and CS techniques.
Our approach can be improved by off-line diagnosability analysis, to anticipate the non discriminable states of the system and this will be the purpose of future work.

References

Bayoudh, M., and Travé-Massuyès, L. 2007. Diagnostic base de modèles hybrides, diagnosticabilité du système multimodes sous-jacent. *Congrès EDSYS*.

Bayoudh, M.; Travé-Massuyès, L.; and Olive, X. 2007. Hybrid systems diagnosability from a discrete event abstraction of multimode parity space. In *Internal Report LAAS N06804*.

Cocquempot, V.; Mezyani, T. E.; and Staroswiecki, M. 2004. Fault detection and isolation for hybrid systems using structured parity residuals. *IEEE/IFAC-ASCC: Asian Control Conference*.

Cordier, M.; Dague, P.; Lévy, F.; Montmain, J.; Staroswiecki, M.; and Travé-Massuyès, L. 2004. Conflicts versus analytical redundancy relations : A comparative analysis of the model-based diagnostic approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man and Cybernetics, Part B*. 34(52163-2177).

Henzinger, T. 1996. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, 278–292.

Hofbaur, M., and Williams, B. 2004. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*. 34(5):2178–2191.

Pencolé, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, 43–47.

Sampath, M.; Sengputa, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketsis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40:1555–1575.

Staroswiecki, M., and Comtet-Varga, G. 2001. Analytic redundancy relations for fault detection and isolation in algebraic dynamic systems. *Automatica* 37:687–699.

Travé-Massuyès, L.; Escobet, T.; Spanache, S.; and Olive, X. 2004. Diagnosability analysis based on component supported analytical redundancy relations. *Rapport LAAS N04080, To appear in IEEE Transactions on Systems, Man and Cybernetics, Part A*.

Chronicle modelling using automata and colored Petri nets

Olivier Bertrand and **Patrice Carle**
ONERA-DPRS, Chatillon, France
(olivier.bertrand,patrice.carle)@onera.fr

Christine Choppy
Université Paris 13, Villetaneuse, France
Christine.Choppy@lipn.univ-paris13.fr

Abstract

One key issue in system simulation is to detect undesired or dangerous activities. An activity may be modelled by a chronicle described by a combination of events occurrences.

We present in this paper how chronicle recognition can be achieved by modelling with different kinds of automata including colored Petri nets.

The relationships between the events may be logical or temporal. As regards the logical relationships we take into account the disjunction and conjunction, and for the temporal relationships we consider the sequence and the absence of an event.

We illustrate our different models with the recognition of some chronicles within different flow of events.

Introduction

One key issue in system simulation is to detect undesired or dangerous activities. An activity may be modelled by a chronicle described by a combination of events occurrences. When these events are identified within a flow of events and in the described combination, then this activity is detected. It is thus necessary to model the chronicle recognition in a general way so as to encompass the different possible combinations.

A chronicle describes relationships between the events of a sequence ordered with respect to time. The goal is to identify the chronicle schema within an observed event flow (where events are ordered and time-stamped). The chronicle identification is achieved through the matching between events of the flow and events in the chronicle description. In addition, it may be of interest to save the piece of information stating which events in the flow contributed to the chronicle recognition, because it may help to find which are the causes of the observed events.

The relationships between the events may be logical or temporal. The logical relationships we consider here are the conjunction of two events (e.g. A and B) where no ordering is involved, and the disjunction (A or B). As regards the temporal relationships, we consider here the sequence (e.g. event A followed by event B) and the absence of an event (e.g. event C should not occur between A and B). Let us note that an event absence does not correspond to a negation since its scope is bounded in time. In the same way, a sequence may also be bounded in time.

In this work, we propose different models for chronicle recognition using automata or Petri nets. We illustrate the different models with the recognition (in a flow of events) of a chronicle Ch described by the sequence of the event A followed by the event B where no event C should occur between A and B. One of the issues in chronicle recognition is to find *all* instances of the studied chronicle(s). For instance Figure 1 shows the five occurrences of the chronicle Ch to be identified in an event flow A A B A B, while the chronicle Ch would be identified only once in an event flow A A C B A B (because the occurrence of C before the first B cancels the first four instances found in Figure 1).

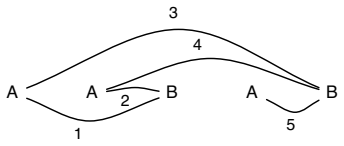


Figure 1: Chronicle Ch has five occurrences in A A B A B

In the section devoted to chronicle recognition modelling through finite automata, we first show the limitations of standard finite state automata, and then show how we can use either counter automata or duplicating automata to overcome the finite state automata limitations. We then show how colored Petri nets may be used for the chronicle recognition. Finally, we show how our approach can be used also to the logical relationships (disjunction and conjunction), and we discuss related works and future intended work.

Chronicle modelling through finite automata

Standard automata

In this section, modelling with finite state automata is considered, where a finite state automaton is defined by a finite set of states, a finite set of transition labels, a set of transitions, and a distinguished initial state (Bérard *et al.* 2001). A finite state automaton recognizes regular expressions, and chronicle recognition differs from regular expression recognition since multiple occurrences of a chronicle should be

identified. Thus, the first automaton in Figure 4 recognises only once the chronicle Ch. In the following sections, we shall present other kinds of finite state automata designed to recognize the multiple occurrences of the chronicle Ch.

Let us, however, explain the behaviour of the automata in Figure 2 where the state wA is waiting for the event A to occur, the state wBC waits for events B or C, in the state S the chronicle is successfully identified, while the state F fails to recognize it (because of the occurrence of event C).

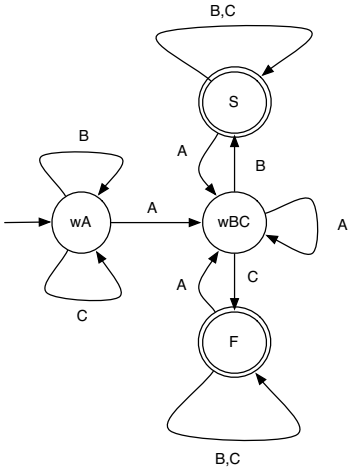


Figure 2: A finite state automaton for the chronicle Ch recognition (A followed by B without C between A and B).

If the input event flow is A C B A B, the chronicle should be recognized once, which is the case as shown by the following automaton execution

$\rightarrow$ wA $\stackrel{A}{\rightarrow}$ wBC $\stackrel{C}{\rightarrow}$ F $\stackrel{B}{\rightarrow}$ F $\stackrel{A}{\rightarrow}$ wBC $\stackrel{B}{\rightarrow}$ S .

If the input event flow is A A B, the chronicle should be recognized twice, but the automaton only finds one occurrence as shown in the execution below

$\rightarrow$ wA $\stackrel{A}{\rightarrow}$ wBC $\stackrel{A}{\rightarrow}$ wBC $\stackrel{B}{\rightarrow}$ S .

With a finite state automaton, we could model the event sequence (A followed by B) and the absence (here of C). However, we cannot recognize the multiple occurrences of a chronicle. This problem will be fixed with the use of counter automata.

Counter automata

Here, integer variables may be introduced as counters which are updated at the transition firings. It is also possible to use these counters in transition guards.

In the automaton of Figure 3, the variable Na is used to count the number of events A, and the variable NCh to count the number of Ch instances that are recognized. With the input event flow A A B, the chronicle should be recognized twice, which is the case in the following execution.

$\rightarrow$	wA	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{B}{\rightarrow}$	S
	Na=0		Na=1		Na=2		Na=2
	NCh=0		NCh=0		NCh=0		NCh=2

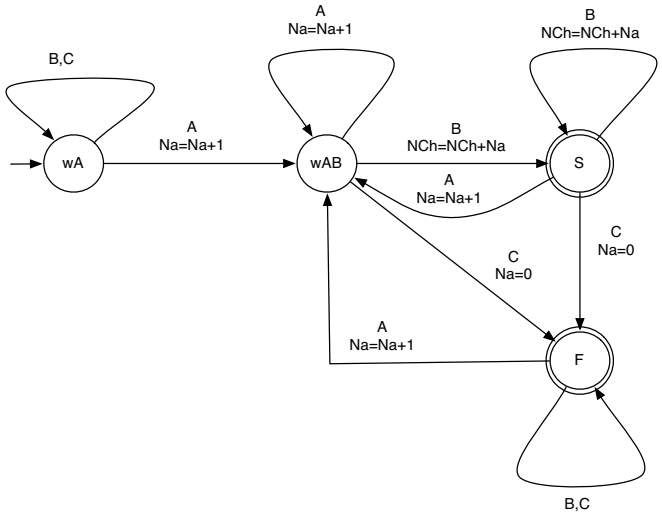


Figure 3: A counter automaton for the chronicle Ch recognition (A followed by B without C between A and B).

With the input event flow A A B A B, the chronicle should be recognized five times (as shown in Figure 1), which is the case in the following execution.

$\rightarrow$	wA	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{B}{\rightarrow}$	S	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{B}{\rightarrow}$ S
	Na=0		Na=1		Na=2		Na=2		Na=2	Na=3
	NCh=0		NCh=0		NCh=0		NCh=2		NCh=2	NCh=5

With the input event flow A C B A B, the chronicle should be recognized only once (note that, after C occurred, Na is set back to 0).

$\rightarrow$	wA	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{C}{\rightarrow}$	F	$\stackrel{B}{\rightarrow}$	F	$\stackrel{A}{\rightarrow}$	wBC	$\stackrel{B}{\rightarrow}$ S
	Na=0		Na=1		Na=0		Na=0		Na=1	Na=1
	NCh=0		NCh=0		NCh=0		NCh=0		NCh=0	NCh=1

With a counter automaton, we could model the event sequence (A followed by B) and the absence (here of C), and we could recognize the multiple occurrences of a chronicle (and count how many).

Duplicating automata

Duplication of automata goes back to the self duplicating machine of Von Neuman, it was then used in different domains, and in cellular automata. Here it is inspired by object oriented languages, with the idea that the automaton corresponds to a class. For each chronicle partial recognition (that is at each state change) an automaton instance is generated with the current state of the automaton. With this framewok, it is also possible to explicitely distinguish the different occurrences of an event (e.g. A:1, A:2 the first and second occurrences of event A, respectively). In order to explain the behaviour, let us consider, with the same chronicle Ch to be recognised, the automaton in Figure 4. It is simpler than the one in Figure 2 because transitions from the final states S and F are removed.

The execution initial state is wA. When an event A:1 (that is the first occurrence of A) occurs, the automaton is duplicated as shown in Figure 5 where the arrow indicates the

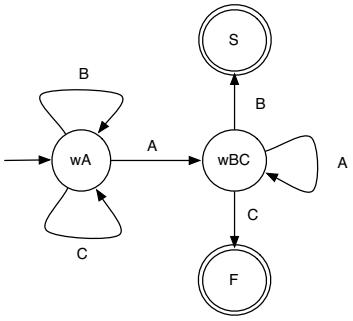


Figure 4: A finite state automata

automaton state change, while the dashed arrow denotes the duplication. The duplication occurs upon each state change, and generates an instance of the automaton in the state before the change occurred (here in state wA). Both arrows are labelled with the event occurrence. Two automata are now in parallel execution.

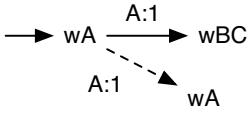


Figure 5: Duplicated automaton after event A:1

When another event A:2 occurs, the first automaton stays in the same state, while the second one changes state and duplicates, as shown in Figure 6.

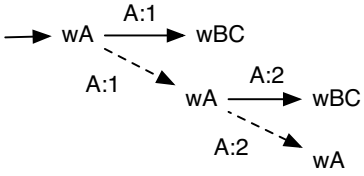


Figure 6: Duplicated automaton after event A:2

Now an event B:1 occurs and causes both automata to change state (and reach the success state since the chronicle Ch was recognized) and duplicate, as shown in Figure 7. The chronicle was recognized twice, through A:1 B:1, and A:2 B:1. Thus event B:1 is used in both automata that were waiting for an event B, and are now in the final state S.

When an event C occurs as in the event flow A C B A B in Figure 8, the first automaton instance is in the failure state, while the second automaton instance will be matched with the last event B, and the chronicle is recognized once. Let us

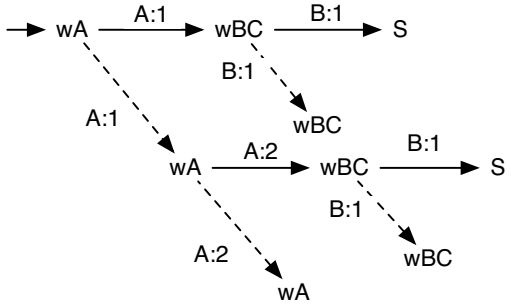


Figure 7: Duplicated automaton after event B:1

note that the automaton on the first line was not duplicated because it reached the failure state (indeed a duplication at this point, i.e. in state wBC, would allow a recognition of Ch if an event B occurred). The first event B (occurring after C) does not induce a change of state in the second line automaton, thus this is not taken into account (and does not contribute to any chronicle recognition).

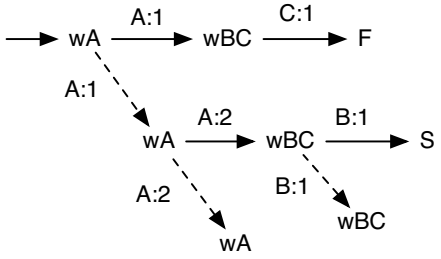


Figure 8: Duplicated automaton after the event flow A C B A B

Thus duplicated automata may be used to recognize all chronicle instances, and also to save the information about which event occurrence contributed to the recognition.

We showed that chronicle recognition can be satisfactorily modelled using either counter automata or duplicating automata which both recognize all instances of a chronicle in a given flow of events, while taking into account the absence of an event (event C in our example). Our modelling with duplicating automata also provides the information about which events occurrences (in our example, the two occurrences of event A, A:1, A:2) contributed to the chronicle recognition.

Chronicle modelling by colored Petri nets

A Petri net is a kind of automaton which state is denoted by the tokens at its different places, and where the different possible transitions are also displayed. In colored Petri nets (Jensen 1995), tokens are multiset of declared datatypes

values. For our modelling of chronicles, we declare a type `Event` (that enumerates the different possible events), a type `Chronicle` (a list of events which can be matched with a chronicle), and a type `MyList` that is a list of the (partially) recognized chronicles. For our example, we have the following types:

```
type Event= with A | B | C;
type Chronicle=list Event;
type MyList = list Chronicle;
```

and the following variables are used within the colored nets

```
var la, fail, instance: MyList;
```

The list type is predefined together with the :: (append), ^^ (concat), hd (head) and tl (tail) functions, and [] denotes the empty list. We define the `MyAdd` function so as to add a given event to each list representing a partially recognized chronicle.

```
fun MyAdd(e:Event,t:MyList)=
if (t!=[])
then
((hd t)^^[e])::MyAdd(e,(tl t))
else []
```

Thus `MyAdd(B,[[A],[A]])=[[A,B],[A,B]]`. In the colored Petri net in Figure 9, the place P is typed with MyList (to be placed below the place oval), and initially contains the empty list [].

Let us first show how colored nets may be used to model the recognition of an event occurrence.

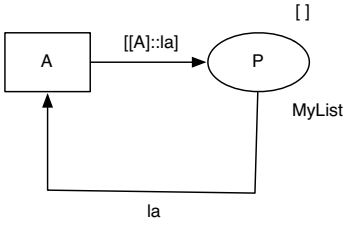


Figure 9: A colored net for an event recognition

The colored net in Figure 9 models the recognition of an event A. The token in place P contains the list of recognized events, which is initially the empty list []. The initial marking is thus denoted by the vector $[\ []\ ]$.

The occurrence of an event A is modelled by the firing of the transition A, and the token in place P is modified by appending the list [A] to it (more precisely, the place P token [] matches the transition input arc variable la and [A] is added to it on the transition output arc). The following execution shows the evolution of the net marking after firing twice the transition A. The resulting marking denotes that A was recognized twice.

$[\ []\ ] \xrightarrow{A} [\ [[A]]\ ] \xrightarrow{A} [\ [[A]\,,[A]]\ ]$

Let us note that, using a function slightly more elaborate than append (::), it would be possible to number the different occurrences of A (as shown with the duplicating automata).

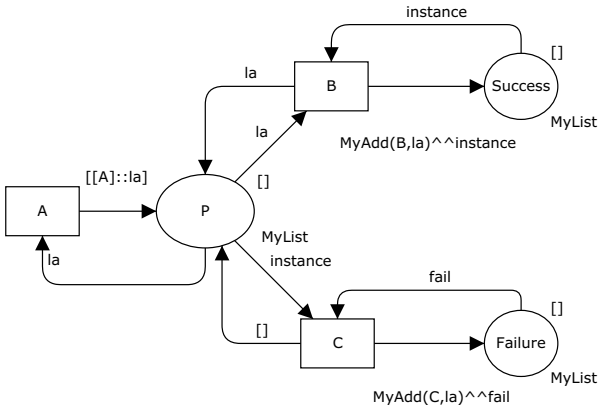


Figure 10: Colored net for the chronicle Ch recognition

Let us now describe the colored net in Figure 10 that models the chronicle Ch recognition. The transition A (connected to the place P) is as in Figure 9 above. The transition B models the occurrence of the event B, its input places are both place P (containing event A occurrences), and place Success (containing the occurrences of [A,B] matching the chronicle Ch). Firing the transition B will result in "appending" (in a specific way achieved using the MyAdd function) all instances of [A,B] to the place Success token, as shown in the following execution flow with event flow A A B.

$$\begin{array}{c}\text{P}\\ \text{Success}\\ \text{Failure}\end{array}\begin{bmatrix}[]\\[]\\[]\end{bmatrix}\xrightarrow{A}\begin{bmatrix}[[A]]\\[]\\[]\end{bmatrix}\xrightarrow{A}\begin{bmatrix}[[A]\,,[A]]\\[]\\[]\end{bmatrix}\xrightarrow{B}$$

$$\begin{bmatrix}[[A]\,,[A]]\\[[A,B]\,,[A,B]]\\[]\end{bmatrix}$$

Let us note that the MyAdd function requires that la (the list of A events) is not empty, otherwise it returns the empty list. In this way, if an event B occurs before an event A, this will not lead to any wrong recognition. Another point is that firing transition B puts back in place P the list of events A that occurred so far, which are then available for any other instance of the chronicle. Transition C behaves in a similar way, yielding to the Failure place, but it cancels all preceding events A by putting back the empty list [] in place P, as for instance in the execution with the event flow A C B A B.

$$\begin{array}{c}\text{P}\\ \text{Success}\\ \text{Failure}\end{array}\begin{bmatrix}[]\\[]\\[]\end{bmatrix}\xrightarrow{A}\begin{bmatrix}[[A]]\\[]\\[]\end{bmatrix}\xrightarrow{C}\begin{bmatrix}[]\\[]\\[[A,C]]\end{bmatrix}\xrightarrow{B}$$

$$\begin{bmatrix}[]\\[]\\[[A,C]]\end{bmatrix}\xrightarrow{A}\begin{bmatrix}[[A]]\\[]\\[[A,C]]\end{bmatrix}\xrightarrow{B}\begin{bmatrix}[[A]]\\[[A,B]]\\[[A,C]]\end{bmatrix}$$

We showed that chronicle recognition can be satisfactorily modelled using colored Petri nets, to take into account the sequence of events, the absence, and also to recognize

all instances of a chronicle in a given flow of events While it would have blurred the readability (and so we did not add it), the function `MyAdd` could have been specialized to number and distinguish the different events that contribute to the recognition of the chronicle instances.

Modelling disjunction and conjunction

The same approach is used to model the recognition of chronicles with logical relationships between events, such as the disjunction "A | B" (with the meaning A or B), and the conjunction "A & B" (with the meaning that both events should occur, but no temporal ordering is enforced).

Figure 11 displays the counter automaton for the recognition of A | B, the duplicating automaton is obtained by removing the counter expressions as well as the state S loop transitions.

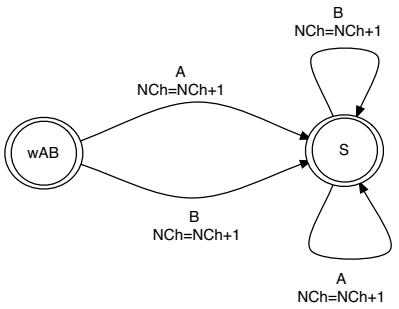


Figure 11: A counter automaton for A | B.

The colored Petri net for the same chronicle A | B is displayed in Figure 12.

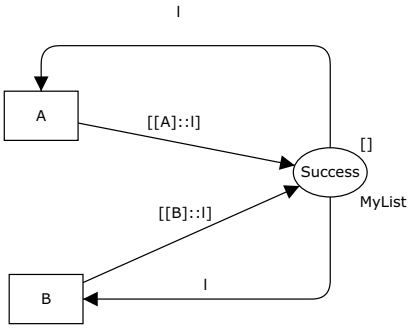


Figure 12: Colored net for the chronicle A | B recognition

Figure 13 displays the counter automaton for the recognition of A & B, the duplicating automaton is obtained by removing the counter expressions as well as the state S loop transitions.

The colored Petri net for the same chronicle A & B is displayed in Figure 14, a new function is defined `MyAddAnd` that is similar to `MyAdd`.

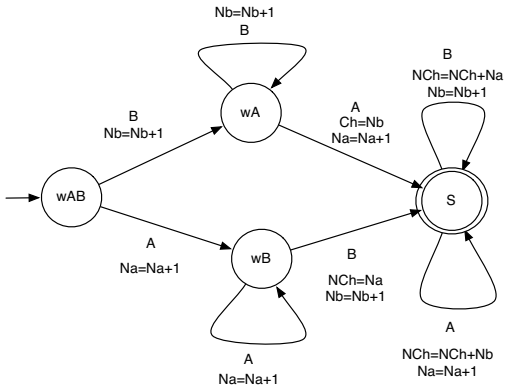


Figure 13: A counter automaton for A & B.

```
fun MyAddAnd(e:Event,t:MyList,
             evt:Event) =
 if t!=[] then
  if ((tl t)!=[]) then
   if (hd(hd(t)))=evt then
    ((hd t)^^[e])::MyAddAnd(e,(tl t),evt)
    else MyAddAnd(e,(tl t),evt)
   else if #1(hd(hd(t)))=evt
        then [(hd t)^^[e]]
        else []
  else []
```

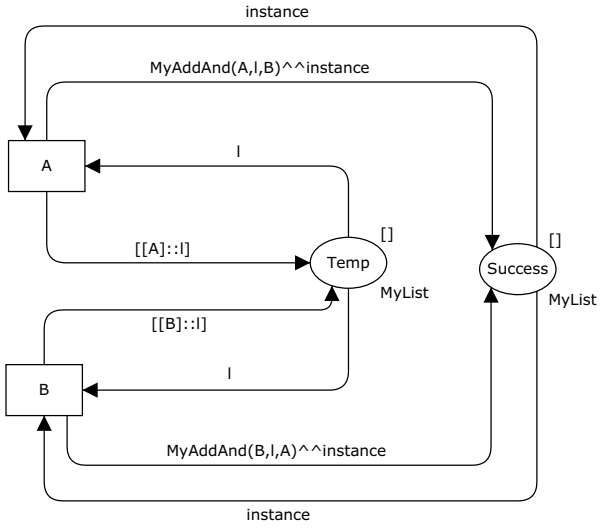


Figure 14: Colored net for the chronicle A & B recognition

Thus we could also model the conjunction and the disjunction relationships and the correponding chronicle recognition using counter automata, duplicating automata or colored Petri nets.

Conclusions, related works and perspectives

In this paper, we compared four automaton formalisms (finite state automata, counter automata, duplicating automata, and colored Petri nets) in their use for modelling chronicles recognition. The relationships we considered so far for the chronicle events are temporal (the sequence, and the absence), and logical (the conjunction and the disjunction). Let us note that our approach encompasses the absence of chronicles (and not only of an event, although, for simplicity sake, this was the example we used). While all these relationships could be modelled with the different formalisms we used, multiple occurrences of a chronicle could not be detected using standard finite state automata. Another point of interest is to be able to distinguish the different chronicle occurrences, with the information of which event occurrence contributed to which chronicle occurrence. This could not be achieved with counter automata while it could be done using duplicating automata or colored Petri nets. While the modelling with the duplicating automata seems lighter, the colored Petri nets could be developed with the CPN Tools environment[1], which provides tools for editing, simulating and analysing. The analysis of properties (e.g. reachability, ...) is achieved through the generated state space.

Let us stress that we could express the chronicle absence in this work while to our knowledge this was not achieved in previous works.

In another approach (Boufaied 2003; Boufaied, Subias, & Combacau 2002) chronicle recognition is modelled through timed place transition nets. A. Boufaied (Boufaied 2003) duplicates the nets upon an event recognition in order to recognize all chronicle instances. Using timed nets one can express timed constraints.

C. Dousson (Dousson 1994; Dousson & Le Maigat 2006), and M. Ornato and P. Carle (Carle *et al.* 1998) (Chronicle Recognition System/ONERA) introduced the notion of a chronicle language together with an associated recognition system. M.-O. Cordier *et al.* (Cordier *et al.* 2007) show how chronicles can be extended to take into account decentralised diagnostic for web services.

Petri nets of various kinds were used for dynamic systems diagnosis. In (Genc & Lafortune 2007), place-bordered Petri nets (with associated diagnosers) are used to model the components of modular dynamic systems, and fault diagnosis is achieved through a distributed algorithm. In (Jard, Chatain, & Bourhis 2005), unfolding timed Petri nets is used for the diagnosis of distributed systems.

As regards hybrid systems monitoring, (Lesire-Cabaniols & Tessier 2007) use particle Petri nets with an associated estimation process and detect inconsistent situations in a pilot activity scenario.

In (Bertrand, Carle, & Choppy 2007), we show how we started to perform off line analysis of distributed HLA simulations using chronicles. We plan to model in the future other relationships between events such as timed delays and timed relationships.

Acknowledgements We would like to thank the anonymous referees for their helpful comments and remarks.

[1] http://wiki.daimi.au.dk/cpntools/cpntools.wiki

References

Bérard, B.; Bidoit, M.; Finkel, A.; Laroussinie, F.; Petit, A.; Petrucci, L.; and Schnoebelen, Ph. 2001. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer.

Bertrand, O.; Carle, P.; and Choppy, C. 2007. Vers une exploitation des simulations distribuées par les chroniques. In *8e Rencontres nationales des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 07)*.

Boufaied, A.; Subias, A.; and Combacau, M. 2002. Chronicle modeling by Petri nets for distributed detection of process failures. In Press, I. C. S., ed., *Second IEEE International Conference on Systems, Man and Cybernetics (SMC'02)*. IEEE.

Boufaied, A. 2003. *Contribution à la surveillance distribuée des systèmes à événements discrets complexes*. Ph.D. Dissertation, Université Paul Sabatier.

Carle, P.; Benhamou, P.; Dolbeau, F.-X.; and Ornato, M. 1998. La reconnaissance d'intentions comme dynamique des organisations. In *6èmes Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98)*.

Cordier, M.-O.; Le Guillou, X.; Robin, S.; Roz, L.; and Vidal, T. 2007. Distributed Chronicles for On-line Diagnosis of Web Services. In *The 18th International Workshop on Principles of Diagnosis (DX-07)*.

Dousson, C., and Le Maigat, P. 2006. Improvement of chronicle-based monitoring using temporal focalization and hierarchization. In *DX'06, 17th International Workshop on Principles of Diagnosis*.

Dousson, C. 1994. *Suivi d'évolution et reconnaissance de chroniques*. Ph.D. Dissertation, Université Paul Sabatier.

Genc, S., and Lafortune, S. 2007. Distributed Diagnosis of Place-Bordered Petri Nets. *IEEE Transactions on Automation Science and Engineering* 4(2):206–219.

Jard, C.; Chatain, T.; and Bourhis, P. 2005. Diagnostic temporel dans les systèmes répartis à l'aide des dépliages des réseaux de Petri. *Journal Européen des Systèmes Automatisés (JESA)* 39(1-2-3):351–366.

Jensen, K. 1995. *Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 1, vol. 2 et vol. 3*. London, UK: Springer-Verlag.

Lesire-Cabaniols, C., and Tessier, C. 2007. Particle Petri net-based estimation in hybrid systems to detect inconsistencies. In *DCDS'07, 1st IFAC Workshop on Dependable Control of Discrete Systems*.

Improving Diagnostic Accuracy by Blending Probabilities: Some Initial Experiments

Stephyn G. W. Butcher **John W. Sheppard**

Numerical Intelligent Systems Laboratory
Department of Computer Science
The Johns Hopkins University
3400 N. Charles Street
Baltimore, Maryland 21218
{sbutche2,jsheppa2}@jhu.edu

Abstract

Inspired by the impending availability of asset specific data on several US Department of Defense programs, in a previous paper we looked at the possibility that a set of Bayesian diagnostic models constructed from asset specific data would outperform a single Bayesian diagnostic model constructed from all of the data. There were situations where a set of asset-specific classifiers was superior to a single composite classifier but it wasn't universally the case. The hypothesis in this paper is that a blended classifier can be constructed to take advantage of the best of both worlds: have a composite classifier's accuracy when its individual accuracy was greater and have an asset specific classifier's accuracy when its accuracy was greater. Our experiments suggest that a split classifier—one that uses asset-specific data to estimate priors and composite data to estimate the likelihoods—can more correctly represent the distributions in the underlying diagnostic problem.

Introduction

In a previous paper, we investigated the possible advantages and disadvantages of creating asset specific diagnostic models instead of a single diagnostic model covering all assets (Butcher *et al.* 2006). The theoretical impetus was the general observation that when constructing any classifier, the more closely the distribution of the sample data matches the distribution of the target population, the more accurate the model will be. In terms of Bayesian diagnostics, this suggested that if assets or groups of assets experienced distinct failure patterns, a set of diagnostic models tuned to the individual assets should be more accurate than a single diagnostic model covering all assets. However, such an approach requires that test and maintenance data be tagged with unique identifying information.

The practical impetus for the paper was the incipient availability of the required asset specific data through the Department of Defense's Item Unique Identification (IUID) numbers. When DoD's IUID program is fully implemented, maintenance and testing data that is tracked to specific assets will be readily available, which is exactly what this approach would require.

To test our hypothesis, we performed experiments comparing the accuracy of a single classifier against a set of asset specific classifiers over ranges of data sizes and levels of noise in the data. The main result was that a set of asset-specific classifiers was often better than a single composite classifier especially when noise levels were high and data was sparse. Unfortunately, the hypothesis was not supported universally. Moving in the direction of less noise, it was often the case that asset-specific classifiers were less accurate overall than the composite classifier. There was also a broad middle ground where they were both equally accurate.

The purpose of this paper is to answer a question left previously unanswered: if a set of asset specific classifiers is sometimes better and sometimes worse than a single classifier but just as good the rest of the time, when should each be used?

While the patterns in our results suggested that weighting the two classifiers might be the solution, we rejected applying a simple ensemble approach. Instead we hypothesized that we could *blend* the probabilities within each classifier of the set and be as accurate as either of the original approaches, thus attaining the best results of each. Surprisingly, our results show that a set of classifiers using blended probabilities was able to *outperform* both the original asset specific set and the composite classifiers. This result led us to identify a simpler blending scheme for the constituent probabilities based on the structure of the problem. We leave for future work problems that may require the more complicated blending scheme.

The plan of the paper is as follows. The first section will give some background on the Bayesian approach to diagnostics. The second section will discuss some related work in ensemble methods. The third section describes the experimental design. The fourth section reviews the results of our previous experiments. In the fifth section we look at our current results. In the final section we summarize our findings and make suggestions for future work.

Bayesian Approaches to Diagnostics

Developing system models for diagnosis is complex and often depends on a detailed understanding of system performance and test engineering. Learning diagnostic models from field maintenance data offers considerable potential to develop or refine diagnostics for fielded systems. Simulation can also be used to generate data for purposes of learning. Several approaches exist for learning such models including case based reasoning, decision tree induction, neural networks, and Bayesian methods. In previous work, we showed how diagnosis and classification are related through the D-Matrix Model (Sheppard and Butcher 2007). We decided to investigate Bayesian methods to diagnosis because they derive models based on sound mathematical principles, they can adapt easily as more data is obtained, and they have been demonstrated empirically to perform well on a broad range of classification problems.

Previously, Sheppard and Kaufman (2005) provided a detailed derivation of a simple model for Bayesian diagnosis. We have also demonstrated how both the Naïve Bayes Classified (NBC) and the Tree-Augmented Bayesian network (TAN) perform on a small sample of IUID-enabled data for a US Navy weapon system (Sheppard *et al*. 2006). For the purposes of this paper, the NBC will be sufficient for the experiments we are going to perform.

The primary assumption for NBC is that the evidence variables in the network (i.e., the tests) are conditionally independent of each other given the class (i.e., diagnosis). Let us define our diagnostic networks as if they contain only one diagnosis variable with n possible values (corresponding to each of the diagnostic conclusions D_i). Thus, we will apply a simple network structure corresponding to the form shown in Figure 1. Note that this structure can be modified where there is a separate Boolean node D_i for each diagnosis rather than a single composite diagnosis node. This leads to the so-called naïve Bayes "multi-net" (Friedman *et al.* 1997; Duda *et al.* 2001).

Under the naïve Bayes model, we attempt to find the specific diagnosis that maximizes the *a posteriori* probability of the diagnosis given the set of observations (i.e., test results). Let $o(T_i)$ be the discrete outcome (e.g., PASS or FAIL) for some test T_i). Then

$$D = \arg\max_{D_i \in \mathbf{D}} P(D_i \mid o(T_1),\ldots,o(T_n)) = \arg\max_{D_i \in \mathbf{D}} P(o(T_1),\ldots,o(T_n) \mid D_i)P(D_i).$$

Unfortunately, the problem remains that the size of the joint distribution over the tests is exponential in the number of tests. But under the naïve Bayes assumption, we can simplify the classification rule to the following:

$$D = \arg\max_{D_i \in \mathbf{D}} P(D_i)\prod_{j=1}^{n} P(o(T_j) \mid D_i).$$

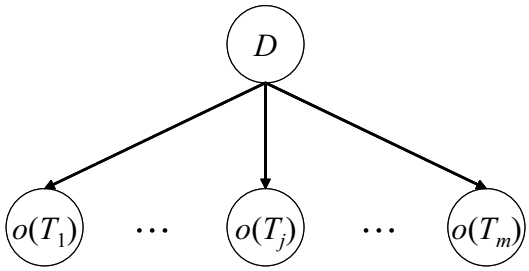


Figure 1. Bayesian diagnostic model

Given a set of training data mapping test results to faults detected, we can "learn" and NBC by observing that $P(D_i)$ is the frequency of occurrence of a particular fault in the data set. Similarly, $P(o(T_j) \mid D_i)$ is the frequency of test outcome $o(T_j)$ considering only the particular diagnosis D_i. What is remarkable about this simple model is the considerable effectiveness it has demonstrated in numerous experiments and implementations (Langley *et al.* 1992).

An important ramification of the NBC rule is that if any $P(o(T_j) \mid D_i)$ should happen to be zero then the entire value of the expression for that particular D_i zeroes out. This is not generally what we want, especially if we have learned our network from sparse training data. To prevent the classification rule from "breaking", the typical solution is to use a default estimate of the likelihoods. Although we will have more to say about this later, the approach we use is an *m*-estimate calculated as follows (Mitchell 1997):

$$P(o(T_i) \mid D_j) = \frac{n_c + mp}{n + m}$$

where n_c is the number of instances in the data pairing particular values for $o(T_i)$ and D_j, n is the total number of instances in the data corresponding to diagnosis D_j, p is a prior estimate for the probability, and m is the number of "virtual" examples in the data.

In spite of the relative accuracy of NBC, we note that a simple trained classifier is unlikely to be sufficient by itself to accurately diagnose faults. Accurate diagnosis generally requires a model created initially by experts and matured as more data is acquired. The *full* diagnostic problem will probably only be solved using classifiers with other types of diagnostic models (Wilmering and Sheppard 2007).

Related Work

One approach to combining models to improve diagnostic accuracy is through the use of "ensemble methods." Ensemble methods seek to improve accuracy by combining recommendations from multiple classifiers (Polikar 2006). Ensemble methods vary widely and include, for example, examples bagging, boosting, and mixtures of experts.

Bagging normally involves the creation a set of classifiers by using bootstrapping to resample the available

data. Boosting involves creating successive classifiers trained on the mistakes of the previous classifier. Both approaches have been used in classifiers used for diagnostics (Hu *et al.* 2004; Li *et al.* 2005). Mixtures of experts create a meta-classifier that combines the results of simpler classifiers and have been successfully used with Bayesian approaches to classification (Titsias and Likas 2000; Bishop and Svensen 2003).

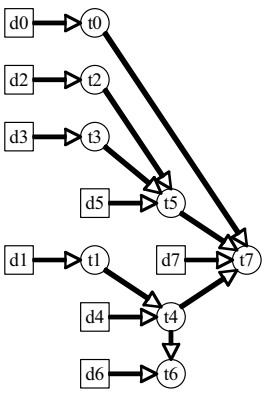
Our research differs from typical ensemble methods in a number of ways. First, while we create a set of classifiers, each classifier is tied to a specific asset. There is no voting because the correct classifier can be determined by context. Second, when creating the classifiers, we apply "blending" at a lower level of abstraction than at the level of the classification results. Although we emphasize the goal of obtaining the best accuracy of either the asset-specific or composite classifiers, we seek to achieve this by combining asset-specific and composite data to estimate the probabilities for each asset-specific classifier.

Experimental Design

To test our hypotheses, we first generated synthetic data. We use a hypothetical system consisting of eight components that can be arranged in various ways. Each component is subject to failure and that failure is detected by a combination of eight tests that can either pass or fail. Depending on how the components are arranged, the diagnostic characteristics of each system are captured by a corresponding D-Matrix (Simpson and Sheppard 1994). Figure 2 shows the arrangement of components and the corresponding D-Matrix for the system used as a basis to generate the data for the experiments in this paper. Each d*i* corresponds to a component that can fail and each t*j* corresponds to a test. In the case of failure, then d*i* corresponds to the diagnosis. Each row in the D-Matrix is a signature relating expected test outcomes (PASS = 0 or FAIL = 1 for each test) to a particular diagnosis.

Because the main purpose of the experiments is to test hypotheses related to how asset specific data might be used to improve diagnostic accuracy, the data needs to be different for different assets. One way to introduce the required differences is to associate a different component failure distribution with each asset. For example, "Asset A" might always have trouble with component 3 (d3). In this case, the probability of d3 failing will be relatively higher than the probability of d0–d2 and d4–d7 failing. For these experiments, we created ten such hypothetical failure distributions each expressing a different behavioral property. The actual distributions used and the properties they represent are described in Butcher *et al.* (2006).

The D-Matrix (system) and component failure distributions (assets) form the foundation for generating the synthetic data. For each data set, *N* data points are generated for each asset using the D-Matrix and a particular fault distribution. For example, if *N* = 100, creating data for "Asset A" (described above) involves creating data that includes seven each of signatures d0–d2 and d4–d7 but 53 of signature d3. This process is repeated for each asset and each *N* value: 25, 50, 100, 250, 500, 1000, 2500 and 5000.



	t0	t1	t2	t3	t4	t5	t6	t7
d0	1	0	0	0	0	0	0	1
d1	0	1	0	0	1	0	1	1
d2	0	0	1	0	0	1	0	1
d3	0	0	0	1	0	1	0	1
d4	0	0	0	0	1	0	1	1
d5	0	0	0	0	0	1	0	1
d6	0	0	0	0	0	0	1	0
d7	0	0	0	0	0	0	0	1

Figure 2. Logic model and D-matrix

At this point, we have a collection of data sets that represent the D-matrix perfectly. However, in the real world, our test results are not likely to originate from clean PASS or FAIL test readings, nor are they always perfect. The actual measurements are subject to varying levels of noise. We also note that an NBC can easily learn the diagnostic concept represented by any D-matrix to 100% accuracy as long as each row in the matrix corresponds to a unique diagnosis because the concept represented is linearly separable (Sheppard and Butcher 2007). In the present case, this will happen whether the data is segregated or aggregated but for these experiments we require at least the possibility that these diverge. So to both inject a degree of realism into the data and to prevent the problem from becoming trivial, we add noise to the data.

Assume we have determined typical "raw" values for a test passing and failing for a specific system and use those values as the means of a Gaussian distribution of equal variance (since we would be using the same measurement device). Based on Bayes decision theory, assuming equal loss, the optimal decision threshold is midway between the two means (Duda *et al.*, 2001). Using different variances, we introduce noise into the data in the following manner. When a test signature is copied into the data set, each test is examined. A random value is generated with the corresponding PASS or FAIL distribution, and the result is compared to the decision threshold[1]. The outcome is then determined based on where the value falls with respect to this threshold. For example, if the result of a particular test is supposed to indicate a PASS (a "0" in the data), a random value is generated with the passing mean and the specified variance. If the resulting value is within the nominal limits,

[1] For these experiments, we assume only a single test limit is applied to determine PASS or FAIL. In fact, this is easily extended to the more realistic case but was deemed unnecessary for these experiments.

the test outcome is kept as a PASS. If it is lower than the nominal limit, the test result is changed to FAIL. Standard deviations (rather than variances) of 0.00 to 0.1 in 0.01 increments are used for a total of 11 different noise distributions.

In Butcher *et al.* (2006), we ran experiments on three systems, ten assets, eight data set sizes, and 11 noise levels for a total of 2,640 data sets. Using this data, NBCs were created for each of ten assets using asset-specific data for a particular system, data set size (N), and noise level as well as a composite NBC using aggregated data. Thus the composite NBC was trained and tested with $10N$ data examples whereas the asset-specific classifiers were each trained with N examples. This comports well with real world experience—if one had data for ten assets and had the option of creating ten classifiers or one aggregate classifier, one would not throw 90% of the data away.

For all experiments, the NBC learning algorithm was repeated with 30 trials using 66% of the data to train the NBC and 34% of the data to test the NBC during each trial. New data was generated for each trial. All random selection was stratified first by system (if necessary) and then by diagnosis (class). The m-estimate was set with p = 0.001% and m = 1. The value of p was set low to make sure that the classification rule doesn't degenerate on the one hand but, on the other hand, the classification is not influenced. Choosing a diagnosis at random breaks all classification ties.

Previous Results

In Butcher *et al.* (2006), we hypothesized that a set of asset specific classifiers where each classifier was trained with its own data would be more accurate overall than a single composite classifier trained using all of the data. However, we didn't necessarily expect this to be true for all N and noise levels. Although the NBC has been shown to train well on relatively few examples (Langley *et al.* 1992), consider an aggregated data set of N = 100 samples. If there are ten assets, this leaves, on average, only ten instances per asset. If there are ten possible diagnoses, this leaves, on average, only one diagnosis per asset. At this point, we may not even be able to determine if assets actually have substantially different fault distributions. Thus we were really addressing two questions: *should* we segregate the data and *when* should we segregate it.

While considering the first question of *should* we segregate, we observed patterns suggesting an answer to the second question of *when*. During the present round of experiments we validated our previous findings and the results at noise levels 0.0, 0.05 and 0.1 are presented in Table 1. The table shows the number of asset-specific classifiers that were at least as accurate as the composite classifier based on a t-test for the difference of means (α = 0.05).

This pattern was typical of results reported by Butcher *et al.* (2006). At low noise levels, both approaches were equally accurate, and this was generally true in the noise range of 0.0 to 0.03. After 0.03, there was a transitional period where some asset-specific classifiers were more accurate than the composite classifier and some were less accurate than the composite classifier. However, once a high enough noise level was reached, usually about 0.06, the accuracy of asset-specific classifiers began to increase relative to the composite classifier. This trend was accelerated when the samples were smaller and the noise levels higher.

Table 1. Asset-specific classifiers vs. composite classifier.

	Noise Level (Std. Dev.)		
N	**0**	**0.05**	**0.1**
25	10	7	7
50	10	5	4
100	10	5	8
250	10	0	10
500	10	1	10
1000	10	2	10
2500	10	8	10
5000	10	8	10

These results were encouraging for asset-specific classifiers in general but because the asset-specific classifiers were not universally superior, we needed a way to decide when to use each approach. However, because the pattern was fairly regular, this suggested that there might be a way to get the best of both worlds: get the accuracy of composite classifiers when they are more accurate and get the accuracy of asset-specific classifiers when they are more accurate. Nevertheless, we were intent on finding an approach that was not *ad hoc*.

New Results

Based on the patterns observed in our prior results, we hypothesized that a data-driven *blended* classifier built from both composite and specific data might achieve a higher level of *overall* accuracy than either previous approach taken singly. Formally, if $C(N,\sigma)$ is classifier *accuracy* as a function of data set size and noise, then we sought an approach with the following as the best case scenario:

$$C_b(N,\sigma) = \max\{C_c(N,\sigma), C_u(N,\sigma)\}$$

where b is blended asset-specific, c is composite and u is unblended asset-specific. As previously discussed, this is not an ensemble approach. There is still a set of asset-specific classifiers being trained, one for each asset. Instead each classifier uses probabilities learned from both asset-specific data and combined data for all assets. The open question was how to blend these two data sources within each asset-specific classifier to achieve the best case scenario.

As previously described, naïve Bayesian classification proceeds by choosing the class that maximizes the product of the prior probability of the diagnosis, $P(D_i)$, and the likelihoods over the tests given the diagnosis, $P(o(T_j) \mid D_i)$.

When determining each probability, the *m*-estimate is used to adjust the calculation as described above. The key components of the *m*-estimate are the values of *m* and *p*, the number of virtual examples and the Bayesian probability estimate of that likelihood. In the absence of better information (which is usually lacking), the *m*-estimate of the likelihood is assumed to take on a constant uniform distribution.

Our approach to blending the data sources leverages the *m*-estimate. We still create a set of asset specific classifiers where each classifier is calculated from asset specific data. The blending comes from applying a modified *m*-estimate calculated using the composite data, $p = P(o(T_j) \mid D_i)$. Additionally, we change the weight of the Bayesian estimate as data set size and noise vary with the aim of matching the patterns we observed in our previous results. Instead of using $m = 1$, we used an *m* that was proportional to *N* and the noise level:

$$m = \frac{k}{\mathrm{var}(o(T_i), D_j)\sqrt[q]{N}} + 1$$

where *k* and *q* are user defined constants. This formula was based on our observation that, keeping noise constant, as *N* increased, we wanted to weight towards the probability calculated from asset specific data and away from the probability calculated from composite data represented by the *m*-estimate. This formula reduces the number of virtual examples, *m*, representing the *composite* based probabilities as *N* increases. With *N* held constant and the noise level increasing, we also wanted to weight more towards the probability calculated from asset specific data and away from the probability calculated from composite data. The formula above fills both requirements, and Table 2 shows some sample calculations of *m* for various *N* and noise levels. In practice, *m* was calculated based on the actual data because the specific noisiness of the data is generally not known *a priori*. As a result, every probability is calculated with its own *m* value. When the variance was zero, it was simply omitted from the formula.

Probabilities calculated from *composite* data were calculated in the usual fashion and used an *m*-estimate with $m = 1$ and $p = 0.001\%$.

Table 2. Example values of *m*

	Noise Level (Std. Dev.)		
N	**0**	**0.05**	**0.1**
25	68400.0	2737.0	685.0
50	38388.7	1536.5	384.9
100	21545.3	862.8	216.4
250	10040.6	402.6	101.4
500	5636.5	226.4	57.3
1000	3163.3	127.5	32.6
2500	1474.6	59.9	15.7
5000	828.0	34.1	9.3

Table 3. Asset specific classifiers vs. composite classifier

	Noise Level (Std. Dev.)		
N	**0**	**0.05**	**0.1**
25	10	5	10
50	10	7	10
100	10	8	10
250	10	9	10
500	10	9	10
1000	10	9	10
2500	10	9	10
5000	10	8	10

Table 4. Comparing asset-specific classifier accuracies

N	**Blended better than Unblended**	**Blended worse than Unblended**	**Blended same as Unblended**
25	3	0	7
50	10	0	0
100	10	0	0
250	5	0	5
500	0	0	10
1000	0	0	10
2500	0	0	10
5000	0	0	10

To test our hypothesis, we constructed a composite classifier ("composite"), a set of asset-specific classifiers ("unblended specific"), and a set of blended asset-specific classifiers ("blended specific") by training and testing them as described in the experimental design section. We used $k = 100$ and $q = 1.2$ in our formula for *m.* The results are shown in Table 3 at noise levels 0.0, 0.05 and 0.1. Similar to Table 1, Table 3 shows the number of blended classifiers out of ten that were at least as accurate as the composite classifier. Cross-referencing with Table 1, we can see that the blended specific classifiers achieved a higher overall level of accuracy as compared to the unblended specific classifiers when compared to the composite classifier. For example, at noise level 0.05 and $N = 250$, no unblended specific classifier was at least as accurate as the composite. However, we can see that nine out of ten blended specific classifiers were at least as accurate as the composite.

Table 4 compares unblended specific and blended specific directly against each other for noise level 0.1. The table shows that at low *N*, the blended specific classifiers were more accurate than the unblended specific classifiers whereas at higher *N*, they have the same accuracy.

These results validated our hypothesis. However, as we looked at the results from a different perspective, we noticed something surprising. Because of how the blended specific classifiers were constructed, we did not contemplate in our hypothesis that they might be *more* accurate than either the composite classifier or unblended specific classifiers in some cases. As previous stated, we supposed that the bounds of accuracy for the blended classifier would be the accuracies of the composite classifier and the appropriate unblended asset-specific classifier. This didn't turn out to be the case. Instead, the

blended classifier was more accurate than either classifier in several cases. Table 5 shows the results for noise level 0.1. There is clearly a pattern, especially at smaller *N*, showing the blended classifier was more accurate on average. This advantage declined as *N* increased.

Table 5. Average accuracies for the classifiers

	Average Percent Accuracy		
N	**Composite**	**Blended Specific**	**Unblended Specific**
25	60.4%	68.6%	62.3%
50	63.4%	68.5%	55.7%
100	64.9%	70.4%	63.0%
250	65.4%	70.6%	68.4%
500	64.9%	70.4%	69.9%
1000	65.4%	70.9%	70.9%
2500	65.5%	71.2%	71.2%
5000	65.4%	71.3%	71.3%

Despite the surprise, we were able to hypothesize an explanation for these results after considering the relationship between the data and the Bayesian model more closely. For a given system, the data reflected two things, different noise levels and different failure distributions—one for each asset. However because the noise level and test signatures where the same for all assets, each classifier was trying to estimate the same *likelihoods*—no matter if it was the composite or one of the unblended asset-specific classifiers. Generally because the composite classifier had 10 times the data, at low *N*, it was better at estimating these likelihoods.

However, the asset-specific classifiers were better at estimating the *prior* probabilities because these depended solely on the failure distributions and each asset had a different and distinctive failure distribution. Whether or not the composite or asset-specific classifier was more accurate depended directly on whether, depending on the noise level and *N*, the prior probabilities or likelihoods were more important for classifier accuracy.

To test this hypothesis, we created a third type of asset-specific classifier, a split asset-specific classifier ("split specific"). Each split specific classifier used data for that asset alone to calculate the prior probabilities and used data aggregated over all assets to calculate the likelihoods.

Comparing the number of split specifics that are as good or better than the composite we find that the split specifics are as good as or better than the composite classifier in the majority of cases. If we compare these results to those we obtained in previous experiments for the unsplit (original) specifics we see that the split specifics out perform the unsplit specifics. As expected, when compared to the blended specifics, the results are comparable (Table 6).

As a result of these three sets of experiments, we are able to answer each of our questions. First, sets of asset specific classifiers can be more accurate than a single composite classifier if the failure distributions are significantly different for the specific assets. However, because of how noise and data sizes interact, it isn't always clear when to use the composite classifier versus the asset specific classifier. Second, the experiments in this paper show that we can construct asset specific classifiers using blended probabilities that do obtain the best of both worlds.

Finally, we also demonstrate in the second set of experiments that if it is the case that the difference between the assets exists solely in their failure distributions, we can create a set of asset specific *split* classifiers that use asset specific data to estimate the priors and the aggregated data to estimate the likelihoods of the Bayesian model. Looking at the specific noise level of 0.1, we find a similar pattern of split specifics being at least as accurate or more accurate than the unsplit specifics (Table 7). When we compare the accuracies of blended and split specific classifiers directly, we find almost no difference in accuracy at all.

Table 6. Split-specific classifiers vs. composite classifier

	Noise Level (Std. Dev.)		
N	**0**	**0.05**	**0.1**
25	10	5	8
50	10	6	9
100	10	7	9
250	10	9	9
500	10	9	9
1000	10	9	9
2500	10	9	9
5000	10	9	9

Table 7. Comparing unsplit and split probabilities

N	**Split better than Unsplit**	**Split worse than Unsplit**	**Split the same as Unsplit**
25	1	2	7
50	1	1	8
100	1	0	9
250	10	0	0
500	10	0	0
1000	10	0	0
2500	7	0	10
5000	0	0	10

It should be noted that although our original research was inspired by the IUID system of the Department of Defense, the approach is valid any time the population of assets can be split into subsets with different failure distributions. For example, different runs or lots of assets may have different failure distributions. As maintenance actions are taken on a population of assets, especially over a protracted period of time, some parts may go out of production to be replaced by components with a different manufacturer. The systems may appear to be the same at the test level but the differing parts may give rise to different failure rates. In short, whenever subsets of a system population begin to exhibit different failure distributions, the accuracy of Bayesian diagnostic models can be improved by using subset (possibly asset) specific data to construct the prior probabilities and aggregate data to construct the likelihoods.

Fortunately, the MTBF for modern systems is generally high. Unfortunately for those attempting to construct diagnostic models, this means failure data is not going to

come all at once or even frequently except for the largest asset populations. It follows that such identifying data should be collected at the start of a program because these differences may not become apparent until much later.

Future Work

The results in this paper were largely driven by our assumptions about what form asset specific behavior might take, how it might show up in the data, and how it could affect building Bayesian diagnostic models. We assumed that asset specific behavior took the form of different failure distributions, and because of the way Bayesian models are built, this affected only a certain part of the model, the prior probabilities.

However, this isn't the only way that asset specific behavior might manifest. From the point of view of Bayesian diagnostics, under what conditions might the likelihood distributions be affected? The likelihoods are the probability of a test result given a particular diagnosis. This would imply that data contain a mixing of asset behavior and testing. How might this come about?

As an example, consider a situation where diagnostic procedures are being prepared for a GPS system that will be installed on a wide variety of aircraft, and that GPS system has a built in health monitor. For example, it could be installed on a C-17 used for long-haul transport with little chance of requiring any extreme maneuvering where we might expect the equipment to function, and fail, as planned. Suppose, however, that the same model GPS system is installed on an F/A-18 that is flying maneuvers in hostile territory. Then more extreme maneuvers may be required, and this could lead to stresses on health monitoring equipment such that the test likelihoods need to shift to reflect greater sensitivity. Finally, consider a GPS system of the same model installed on a high-altitude trainer (e.g., a 747 used by NASA for astronaut training) where it is likely the aircraft will undergo frequent negative G-force maneuvers. A completely different set of test likelihood distributions might appear because of how the measurements are taken.

As a practical illustration of how this might affect learning a Bayesian diagnostic model, consider the D-Matrix for the functional system. Assume that we have three assets operating in environmental conditions such that a health monitor records the following data. For the first asset, data is recorded exactly as would be expected by the D-Matrix (Figure 2). For the second asset, the third test consistently gives the wrong reading. If the expected reading for diagnosis *i* was PASS, then would it read FAIL; FAIL, then PASS. For the third asset, one test in each signature consistently reads falsely in five of the eight test signatures (see Figure 3 for illustrative D-Matrices). In addition to these systematic differences, all readings are subject to varying levels of Gaussian noise (as before).

Because all of this data comes from the same system, the temptation is to aggregate all of the data together to arrive at the best possible model. Assuming a uniform

	t0	t1	t2	t3	t4	t5	t6	t7
d0	1	0	1	0	0	0	0	1
d1	0	1	1	0	1	0	1	1
d2	0	0	0	0	0	1	0	1
d3	0	0	1	1	0	1	0	1
d4	0	0	1	0	1	0	1	1
d5	0	0	1	0	0	1	0	1
d6	0	0	1	0	0	0	1	0
d7	0	0	1	0	0	0	0	1

Environment #1

	t0	t1	t2	t3	t4	t5	t6	t7
d0	1	0	0	0	1	0	0	1
d1	0	1	1	0	1	0	1	1
d2	0	0	1	0	0	1	0	1
d3	0	0	0	1	0	1	0	1
d4	0	1	0	0	1	0	1	1
d5	0	0	0	0	0	1	0	1
d6	0	0	0	1	0	0	1	0
d7	0	0	1	0	0	0	0	1

Environment #2

Figure 3. Environment-dependent D-Matrices

distribution of failures for all three assets, it can easily be shown that aggregation in this case actually harms overall classifier accuracy. In fact, even with no noise, a composite classifier built from equal data from all three assets is only 87.5% accurate compared to 100.0% for the asset-specific classifiers—whether *N* is 25 or 5000.

As the noise level increases, accuracy for every classifier begins to fall but asset-specific classifiers still remain more accurate than the composite classifier. However, after a certain point, around noise level 0.08, there are more ties between the composite classifier and the asset-specific classifiers, especially at low *N*. For example, after running some preliminary experiments with noise level 0.08 and $N = 50$, the accuracy of the composite classifier was 51.5% but the accuracies of the asset-specific classifier for asset No. 1 was 48.4%; No. 2, 52.9%; and No. 3, 53.8%. In addition, none of the differences are statistically significant. Not until $N = 250$ are all of the asset-specific classifiers more accurate than the composite classifier (with statistically significant differences).

This small experiment suggests a number of things. First, in situations where the likelihoods are asset specific, we may have to reverse the "split" classifier so that the priors are estimated from composite data and likelihoods are estimated from asset-specific data. Second, where there are asset specificities arising both from differing failure distributions and test patterns, we may be required to blend probabilities as we did in the first experiment in this paper.

Third, test noise has a completely different effect on asset-specific classifiers when the specificity derives from testing than when the specificity derives from failure distributions. In the latter case, asset specific classifiers were more accurate at high noise and low *N*. In the former case, asset specific classifiers are less accurate at high noise and low *N*. This suggests that the blending will have to happen differently for prior probabilities than for the likelihoods. Finally, constructing, applying and maintaining multiple diagnostic models is relatively more expensive than doing the same for just one. We need to be able find ways to detect the condition and estimate the expected payoff from increased accuracy. We plan to explore all of these subtleties in future research on this topic.

Conclusions

We built on previous work that demonstrated under certain conditions that a set of asset specific classifiers can be more accurate than a composite classifier built from the aggregated data. However, asset specific classifiers were not universally more accurate and this left open the question of when to build a set of asset specific classifiers and when to build a composite classifier.

In this paper, we presented experimental results supporting our hypothesis that a blended approach could improve diagnostic accuracy. The blended approach—while not an ensemble approach—uses aggregated data to create the *m*-estimates and then weights each by a number of virtual examples calculated as a function of both *N* and the level of noise. Although there were still some user-defined variables involved, this avoided an *ad hoc* approach to picking and choosing asset specific classifiers or composite classifiers.

We also showed that because of the structure of the Bayesian model and the kind of asset specificity we were considering, a simpler "split" classifier could be constructed by using asset-specific data to estimate the priors and composite data to estimate the likelihoods.

Finally, we looked briefly into situations where the asset specificity of the data might also affect test signatures in the field and in turn would affect the estimated likelihoods. In this case, we hypothesized that a similar "split" classifier would need to be created for maximal accuracy. We also hypothesized that in the presence of both kinds of asset specificity, we would need to return to the blended approach. All of these problems are left for future work including how we might identify these non-homogeneous situations.

Acknowledgments

The authors would like to thank the anonymous reviewers of the DX07 program committee whose comments greatly improved the quality of the paper. This work was supported, in part, by the US Navy, NAVAIR, PMA 260.

References

Bishop, C. M. and M. Svensen, "Bayesian Hierarchical Mixtures of Experts", *Uncertainty in Artificial Intelligence: Proceedings of the Nineteenth Conference*, 2003.

Butcher S., Sheppard J., Kaufman M., Ha, H. and MacDougall, C. "Experiments in Bayesian Diagnostics with IUID-Enabled Data" IEEE AUTOTESTCON 2006, *Conference Record*, Anheim, CA: September 2006., pp 605-614.

Duda, R., Hart, P., and Stork, D., *Pattern Classification*, New York: John Wiley & Sons, 2001

Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Systems.* Reading, Mass.: Addison-Wesley.

Friedman, N., Geiger, D., and Goldszmidt, M., "Bayesian Network Classifiers," *Machine Learning*, 29:131–163, 1997.

Hu, Z.-H., Y.-G. Li, Y.-Z. Cai and X.-M. Xu, "An Empirical Comparison of Ensemble Classification Algorithms with Support Vector Machines"., *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, Shanghai, China, August 2004, 3520-3523.

Kononenko, I., "Semi-naïve Bayesian Classifier," In Y. Kodratoff (ed.), *Proceedings of the Sixth European Working Session on Learning*, Berlin: Springer-Verlag, 1991, pp. 206–219.

Langley, P., Iba, W., and Thompson, K., "An Analysis of Bayesian Classifiers," *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Mateo, CA: AAAI Press, 1992, pp. 223–228.

Li, Y., Y.-Z. Cai, R.-P. Yin, and X.-M. Xu, "Fault Diagnosis on Support Vector Ensemble", *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou, China, August 2005, 3309-3314.

Mitchell, T. *Machine Learning*, New York: The McGraw-Hill Companies, 1997.

R. Polikar, "Ensemble Based Systems in Decision Making", *IEEE Circuits and Systems Magazine*, 3rd Quarter 2006, 21-54.

Sheppard, J. and Butcher, S., "A Formal Analysis of Fault Diagnosis with D-Matrices" to appear in *Journal of Electronic Testing: Theory and Applications*, 2007.

Sheppard, J. and Kaufman, M., "A Bayesian Approach to Diagnosis and Prognosis Using Built In Test," *IEEE Transactions on Instrumentation and Measurement*, Special Section on Built-In Test, Vol. 54, No. 3, June 2005, pp. 1003–1018

Sheppard, J., Butcher, S., Kaufman, M., and MacDougall, C., "Not-So-Naïve Bayesian Networks and Unique Identification in Developing Advanced Diagnostics," *Proceedings of the IEEE Aerospace Conference*, New York: IEEE Press, March 2006

Simpson, W. and Sheppard, J., *System Test and Diagnosis*, Norwell, MA: Kluwer Academic Publishers, 1994.

Titsias, M. K. and A. Likas, "Mixture of Experts Classification Using a Hierarchical Mixture Model", *Neural Computation* 14, 2000, 2221-2244.

Wilmering, T. J. and Sheppard, J., "Ontologies for Data Mining and Knowledge Discover to Support Diagnostic Maturation", forthcoming, The 18th International Workshop on Principles of Diagnosis (DX-07), Nashville, TN, May 2007.

WS-DIAMOND
Web Services – DIAgnosability, MONitoring and Diagnosis

The Ws-DIAMOND Team*

Abstract

Self-healing software is one of the challenges for IST research. The WS-DIAMOND project aims at making a step in this direction by developing a framework for self-healing Web Services. In particular, the project aims at:

- Defining an framework for self-healing service execution of conversationally complex Web Services, where monitoring, detection and diagnosis of anomalous situations, due to functional or non-functional errors, are carried on and repair/ reconfiguration is performed, thus guaranteeing reliability and availability of Web Services;
- Defining a methodology and tools for service design that guarantee effective and efficient diagnosability/ repairability during execution.

The research builds upon results in different areas such as model-based diagnosis, semantic Web Services, cooperative information systems and Web Service composition. It goes beyond a number of current projects in the area of Service Oriented Computing, which do not consider the monitoring, diagnosing and repairing of Web Services. This paper describes the achievements in the first phase of the project.

Introduction

Ws-DIAMOND is a Strep Project funded by the EU commission under the FET-Open framework. It started in September 2005 and will run until 2008. The project aims at making a step in the direction of Self Healing Web Services. The project, in particular, addresses two different issues concerning Self Healing capabilities:

- **The first goal of WS-Diamond** is to develop a framework for **self-healing** Web Services. A self-healing Web Service is able to **monitor** itself, to **diagnose** the causes of a failure and to **recover** from the failure, where a failure can be either functional, such as the inability to provide a given service, or non-functional, such as a loss of service quality. The focus of WS-Diamond is on composite and conversationally complex Web Services, where *composite* means that the Web Service relies on the integration of various other services, while *conversationally complex* means that during service provision a Web Service needs to carry out a complex interaction with the consumer application.
- **The second goal of WS-Diamond** is to devise guidelines (and tools) for designing services in such a way that they can be easily diagnosed and recovered during their execution.

During the first phase the project concentrated on the first issue and this will be the focus of this paper.

Why is self-healing critical for Web Services? Computing facilities are increasing at a very rapid pace, presenting new forms of interaction, such as in portable and mobile devices, home and business intelligent appliances. This led to the development of complex services to support human activities), in particular creating networks of co-operating services (Web Services). Various complex and intelligent services are becoming available, to support most of our activities, possibly creating a new social environment for interacting and co-operating with other people. The same availability of these services will be critical for allowing us to carry on such activities, in the same way as today we cannot work without having access to corporate or external knowledge sources. The availability and reliability of services will be of paramount importance. Indeed the reliability and availability of software and the possibility of creating self-healing software are recognized as one of the major challenges for IST research in the next years (ISTAG 2004).

Current standards for Web Service design and execution do not include the support for self-healing execution. Simple approaches to perform recovery after failures have been developed but none of them includes approaches to detect the actual causes of the problem (fault diagnosis) and does not support recovery based on fault localization.

WS-DIAMOND proposes the adoption of Model-based diagnosis to tackle this problem. Indeed MBD recently extended its focus from "traditional domains" (artefacts) to new ones, including in particular software systems, communication networks, distributed systems. Particularly significant with respect to this project is the application to

---

(*) WS-DIAMOND (IST-516933) partners: Dipartimento di Informatica – Università di Torino (coordinator), Vrije Universiteit Amsterdam, Diparimento di Elettronica e Informatica Politecnico di Milano, Universitaet Klagenfurt, Université Paris Sud, LAAS Toulouse, IRISA Université Rennes 1, Universitaet Vienna.

WS-DIAMONDers: L. Console, D. Ardagna, L. Ardissono, S. Bocconi, C. Cappiello, M.O. Cordier, P. Dague, K. Drira, J. Eder, G. Friedrich, M.G. Fugini, R. Furnari, A Goy, K. Guennoun, A. Hess, V. , Ivanchenko, X. Le Guillou, M. Lehmann, J. Mangler, Yingmin Li, T. Melliti, S. Modafferi, E. Mussi, Y. Pencole, G. Petrone, B. Pernici, C. Picardi, X. Pucel, S. Robin, L. Rozé, M. Segnan, A. Tahamtan, A Ten Tejie, D. Theseider Dupré, L. Trave Massuyes, F. Van Harmelen, T. Vidal.

Contact address: Luca Console, Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, I-10149 Torino (Italy). Email: luca.console@di.unito.it

WS-DIAMOND is funded by the European Commission, IST, FET-OPEN framework, under grant IST-516933.

software diagnosis in which the same basic technologies have been successfully applied to debug programs (Mateis *et al.* 00; Mayer *et al.* 02) and. component-based software (Grosclaude 04; Peischl et al. 06). The same approach has been applied to the case of Web Services (Mayer et al., 06). The focus of those works is different from our work since they aim at debugging) problems in the composition of Services (orchestration), rather than diagnosing (and repairing) problems arising during service execution.

The paper describes the results of the first phase of WS-DIAMOND. We first introduce an application that will be used as an example in the paper. Then we overview the results we achieved, discussing the assumptions we made in this first phase; we introduce the conceptual framework and architecture we developed for the platform supporting self-healing service execution. In the following sections we discuss the approach we adopted to diagnosis and repair planning. We conclude discussing directions of research in the project.

An application scenario

The main application scenario that we selected is an e-commerce scenario, concerned with a company (called FoodShop) that sells food products on the Web. The company has an online shop (that does not have a physical counterpart) and several warehouses (WH_1, …, WH_n) located in different areas that are responsible for stocking unperishable goods and physically delivering items to customers, depending on the area each customer lives in. Customers interact with the FoodShop Company in order to place their orders, pay the bills and receive their goods. In case of perishable items, that cannot be stocked, or in case of out-of-stock items, the FoodShop Company must interact with one or more suppliers (SUP_1, …, SUP_m).

In the following we describe the business process that brings from the customer order to the parcel delivery, which is of course executed through the cooperation of several services. In particular, in each business process instance we have one instance of the **Shop** service, one instance of a **Warehouse** service, and one or more instances of **Supplier** services. It is important to point out that the business process includes activities that are actually carried out by humans, such as the preparation of the supply package or the physical delivery to the customer. However, we will assume that these activities have an electronic counterpart (a so called wrapper) in the Web Services, whose goal is to track the process execution. For example, when a **Supplier** physically sends supplies to a **Warehouse**, we assume that the person responsible for assembling the supply clicks on a "sent" button on her PC that saves down the shipping note. On the other hand, the person receiving the physical supply clicks on a "received" button on her PC entering in the data shown on the shipping note. Thus we do not make any distinction between electronic and non-electronic activities.

The FoodShop business process

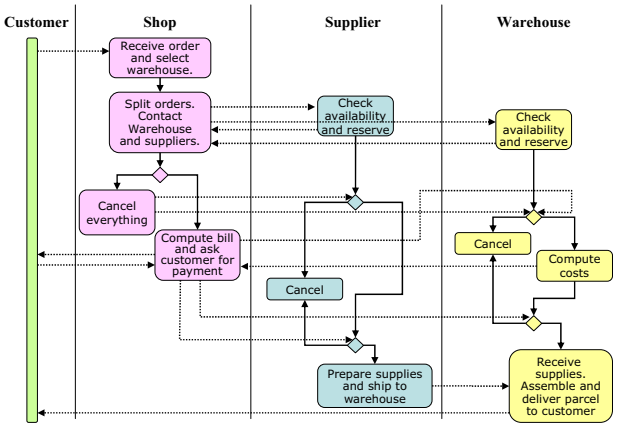


Figure 1: Abstract view of the FoodShop business process.

Figure 1 depicts a high-level view of the business process (some of the details described below are not explicitly shown for the sake of readability). When a customer places an order, the **Shop** service first selects the **Warehouse** that is closest to the customer's address, and that will thus take part in process execution. Ordered items are then split into two categories: perishable (they cannot be stocked, so the warehouse will have order them directly) and unperishable (the warehouse should have them in stock). Perishable items are handled directly by the **Shop**, while unperishable items are handled by the **Warehouse**, although all of them are eventually collected by the **Warehouse** in order to prepare the parcel for the customer. At this point the **Shop** checks whether the ordered items are available, either in the **Warehouse** or from the **Suppliers** (we have not considered items exchanges among different warehouses, in order not to make the example too complicated). If they are, they are temporarily reserved in order to avoid conflicts between several orders. Once the **Shop** receives all the answers on item availability, it can decide whether to give up with the order (again, in order to keep things simple, this happens whenever there is at least one unavailable item) or to proceed. In the former case, all item reservations are canceled and the business process ends. If the order goes on, the **Shop** computes the total cost (items plus shipping) with the aid of the **Warehouse** that provides the shipping costs depending on its distance from the customer location and the size of the order. Then it sends the bill to the customer, that can decide whether to pay or not. If the customer does not pay, all item reservations are canceled and, again, the business process terminates. If the customer pays, then all item reservations are confirmed and all the **Suppliers** (in case of perishable or out-of-stock items) are asked to send their goods to the **Warehouse**. The **Warehouse** will then assemble a package and send it to the customer.

Ws-DIAMOND architecture

Since the overall goal of achieving self healing behaviour is very ambitions, we started by defining in a precise way the requirements we wanted to address in the project and specifically in the first phase.

First of all, we concentrated on the first goal of the project, that is the design and development of a platform for supporting the self healing execution of Web Services. This means that we are concentrating on the problems that occur at run time, while the design issues will be faced in a second phase of the project. A second very general consideration is that we are not considering the issue of debugging a service; in other words we assume that code has been debugged. This led us to defining:

- The types of faults that can occur and we want to diagnose, that is:
  - functional faults and specifically semantic data errors (such as wrong data exchanges, wring data in database, wrong inputs from user, ..);
  - quality of service faults (e.g., delays, quality of data).

  In this paper we will focus on the former.
- The types of observations/tests that can be available to the diagnostic process:
  - alarms raised by services during their executions;
  - data exchanged by services;
  - internal data to a service (we will return later on privacy issues).
- The types of repair/recovery actions that can be performed, such as compensating, re-doing, replacing activities.

In this first phase of the project, moreover we decided to concentrate on orchestrated services (i.e., the case where there is an service orchestrating sub-services, see (Peltz, 2003), even if some of the solutions we are proposing already take into account the case of choreographed services (Peltz, 2003).

The main achievements in the first phase of the project are the following:

- We proposed a Semantic Web Service definition language which includes features that are needed o support the diagnostic process (e.g., observability of some parameters); moreover we started to analyze how these semantic annotations can be learned from service execution logs.
- We extended Web Service execution environments to include features which are useful to support the diagnostic/fault recovery process. In particular, an architecture supporting self-healing service execution has been defined. The architecture provides support for associating a diagnostic service with each service (see below), for gathering observations about service execution (e.g., data exchanged between services) and provides a set of recovery and repair actions. The architecture also includes a monitoring service aimed at identifying Quality of Service and communication problems, and a repair module aimed at supporting the execution of recovery plans on the basis of the diagnostic information.
- We characterized diagnosis and repair for Web Services. In particular we defined a catalogue of faults and possible observations and we proposed an architecture for the surveillance platform (diagnostic service, see below). The core of the platform is a diagnostic problem solver and thus we proposed algorithms for performing the diagnostic process, focusing the attention on functional faults. In particular, the diagnostic architecture is a decentralized one where a diagnoser is associated with each individual service; a supervisor is then associated to the orchestrator service or to the process owner. We defined a communication protocol between local diagnosers and the supervisor, assuming that no knowledge about the internal mechanisms of a Web Service is disclosed by its local diagnoser. A global diagnosis can be computed by the supervisor after exchanging information with local diagnosers (invoking only those that are relevant to solve the specific problem under analysis). The correctness of the algorithms has been proved formally.
- Repair has been then characterized as a planning problem, where the goal is to build the plan of the recovery actions to be performed in order to recovery from errors. The actions are those supported by the execution environment discussed above and involve backtracking the execution of some services, compensating some of the actions that were performed, re-doing activities or replacing faulty activities (or services) with other activities (services).

This led to a set of architectural choices that will be introduced in the next sections.

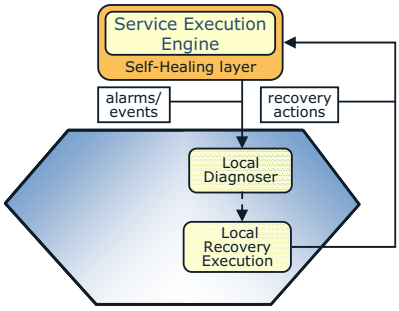


Figure 2: Web Service's DIAMOND

DIAMONDS and Self Healing layer

A DIAMOND is associated with each service and with the orchestrator and is in charge of the enhanced service execution and monitoring, diagnosis and recovery planning and execution. The DIAMOND associated with a basic service is depicted in Figure 2.

The self-healing layer is the set of extensions to the Service Execution Engine that has been designed in the project and that enables monitoring, diagnosis and repair (see below).

The DIAMOND includes diagnosis and repair:

- Alarms and events generated by the service go to the DIAMOND (to Local Diagnoser)
- The Local Diagnoser owns privately the model of the service and is in charge of explaining alarms (events) by either
  - Explaining them with internal faults
  - Blaming other services (from which inputs have been received) as the cause of the problem

  See below for the interaction between Local and Global Diagnosers.
- The Local Recovery Execution module receives recovery actions to be performed from the Global Recovery Planner (see below). It is also admitted that repair actions are selected by the Local Recovery Execution.
- Recovery actions are passed to the Self-healing layer

The DIAMOND associated with the orchestrator is made of two parts (see Figure 3):

- A Global Diagnoser and Recovery Planner (left part)
- A Local Diagnoser and Local Recovery Execution module (right part,)

The latter is meaningful as an orchestrated service may in turn be a sub-service of a higher-level composed service.

The Global Diagnoser interacts with Local Diagnosers to compute a global diagnosis; it does not have access to local models. The diagnosis is computed in a decentralized way. The Recovery Planner operates sequentially after a global diagnosis has been computed. It generates a plan for recovery and passes it to Local Recovery Execution modules.

Self-healing layer

The platform for service execution supports the following features:

- associating a diagnostic service with each service;
- gathering observations about service execution (e.g., data exchanged between services), which are input to diagnosis;
- providing a monitoring service aimed at identifying problems during execution, including Quality of Service and communication problems;
- providing a set of recovery and repair actions (that can be exploited by the repair planner);
- providing a repair module aimed at supporting the execution of recovery plans.

Such a layer can thus be seen as a set of new functionalities on top of a service execution platform (engine)

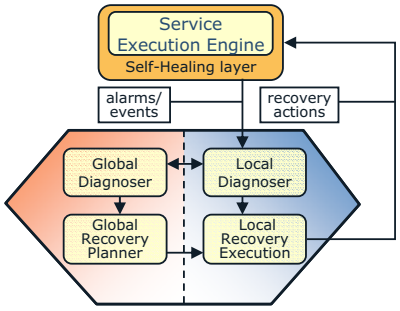


Figure 3: Orchestrator's DIAMOND

Diagnostic approach

The approach we adopted for diagnosis is a model-based one and is based on a decentralized paradigm:

We associate with each basic service a local diagnoser, owning a description of how the service is supposed to work (a model); the role of local diagnosers is to provide the global diagnoser with the information needed in order to identify the causes of a global failure.

We provide a global diagnoser which is able to invoke local diagnosers and relate the pieces of information they provide, in order to reach a diagnosis for the overall complex service. In case the supply chain has several levels, several global diagnosers may form a hierarchy, where a higher level global diagnoser sees the lower level ones as local diagnosers.

We choose to adopt such an approach because this enables to recursively partition Web Services into aggregations of sub-services, hiding the details of the aggregation to higher-level services. This is in accordance with the privacy principle which allows designing services at enterprise level (based on intra-company services) and then use such services in extranets (with other enterprises) and public internets. The global diagnoser service only needs to know the interfaces of local services and share a protocol with local diagnosers. This will mean, in particular, that the model of a service is private to its local diagnoser and needs not be made visible to other local diagnosers or to the global one.

Each local diagnoser interacts with its own Web Service and with the global diagnoser. The global diagnoser interacts only with local diagnosers. More precisely, the interaction follows the pattern described in the following:

During service execution, each local diagnoser receives information from the self healing layer (monitoring the activities carried out by its Web Service, logging the messages it exchanges with the other peers). The diagnoser exploits an internal "observer" component collecting the messages and locally saving them for later inspection. Notice that when a Web Service composes a set of sub-suppliers, the local diagnoser role must be filled by the global diagnoser of the sub-network of cooperating services. On the other hand, an atomic Web Service can

have a basic local diagnoser, that does not need to exploit other lower-level diagnosers in order to do its job.

When a local diagnoser receives an alarm message (denoting a problem in the execution of a service), it starts reasoning about the problem in order to identify its causes, which may be internal to the Web Service or external (erroneous inputs from other services). The diagnoser can do this by exploiting the logged messages. The local diagnoser informs the global diagnoser about the alarm it received and the hypotheses it made on the causes of the error. The global diagnoser starts invoking other local diagnosers and relating the different answers, in order to reach one or more global candidate diagnoses that are consistent with reasoning performed by local diagnosers.

According to the approach discussed above, each local diagnoser needs to have a model of the service in its care. We assume that each Web Service is modeled as a set of inter-related activities which show how the outputs of the service depend on its inputs. The model of a complex service, in particular, specifies a partially ordered set of activities which includes internal operations carried out by the service and invocations of other suppliers (if any).

The model of a Web Service enables diagnostic reasoning to correlate input and output parameters and to know whether an activity carries out some computation that may fail, producing as a consequence an erroneous output. Thus fault localization is performed at the level of the activities of a service.

Symptom information is provided by the presence of alarms, which triggers the diagnostic process; by the absence of other alarms; or by additional test conditions on logged messages introduced for discrimination.

The goal of diagnosis is then to find activities that can be responsible for the alarm, performing discrimination to the purpose of selecting the appropriate recovery action. When an alarm is raised in a Web Service W_i, its local diagnoser A_i receives it. A_i must explain it, Each explanation may ascribe the malfunction to failed internal activities and/or abnormal inputs. It may also make predictions of additional output values, which can be exploited by the global diagnoser in order to validate or reject the hypothesis. When the global diagnoser receives a local explanation from a local diagnoser A_i, it proceeds as follows:

- If a Web Service W_j has been blamed of incorrect outputs, then the global diagnoser can ask its local diagnoser A_j to explain them. A_j can either reject the blame, explain it with an internal failure or blame it on another service that may have sent the wrong input.
- If a fault hypothesis by Ai has provided additional predictions on output values sent to a Web Service W_k, then the global diagnoser can ask A_k to validate the hypothesis by checking whether the predicted symptoms have occurred, or by making further predictions.

Hypotheses are maintained and processed by diagnosers as partial assignments to interface variables and behavior modes of the involved local models. Unassigned variables represent parts of the overall model that have not yet been explored, and possibly do not need to be explored, thus limiting invocations to local diagnosers.

The global diagnoser sends hypotheses to local diagnosers for explanation and/or validation. Local diagnosers explain blames and validate symptoms by providing extensions to partial assignments that assign values to relevant unassigned variables. In particular the global diagnoser exploits a strategy for invoking as less local diagnosers as possible, excluding those which would not contribute to the computation of an overall diagnosis.

Details about the strategies adopted by the global diagnoser, about the communication protocol between global and local diagnosers and about local diagnosers can be found in [Console et al., 2007], where also properties about the correctness of the adopted algorithms are proved.

Example on the FoodShop business process

Let us anlyse how the approach work on the FoodShop example presented above. The faults we consider within the FoodShop business process are (according to the focus of the current project work,) those that:

- affect the correctness of data exchanged between services or between services and customer
- can be detected by looking only at one instance of the business process.

Accordingly, the diagnosis we carry out focuses on finding, within the faulty instance, the last point in execution where data can be safely considered correct. Repair focuses then on planning and executing an alternative portion of the business process that tries to achieve the same goals as the first one without incurring in the same problem. Achieving this my entail substituting a partner (e.g. turning to a different supplier), undoing and redoing some actions (e.g. sending back supplies and receiving correct one), carrying out some compensation activity (e.g. restoring the previous item quantities in the Warehouse database if the wrong quantities were decreased due to a misplaced order), or carrying out some activity-specific repair action (e.g. realigning the Shop database to the Supplier one if it contains obsolete item codes).

Some examples of the faults that we consider within the FoodShop business process are:

- The Shop incorrectly matches ordered items to their codes; since most of the order process works on codes rather than on names this results in the Warehouse not being able to assemble the customer parcel.
- The Shop selects the wrong Warehouse; this can have as effect a higher delivery cost for the customer.
- The Shop does not contact all suppliers, either when reserving (then some items will be missing in the Warehouse when assembling the parcel) or when cancelling or confirming the reservation (these will raise a timeout in the Supplier).
- The Warehouse or the Supplier make a mistake in answering on the availability of the items, so that a non-

available item is declared available; this results in the Warehouse not being able to assemble the parcel.

- The Supplier makes a mistake in saving the supply order, or in physically preparing the supply package; again, the consequence is that the Warehouse will not be able to assemble the customer parcel.

It is easy to see from this list that many faults actually have the same symptoms (namely, that the Warehouse cannot assemble the customer parcel).

We will now see a diagnosis example that shows how some of these faults can be diagnosed by automated reasoning on a distributed model of the business process that includes data dependencies.

Let us assume that the Warehouse service in the example raises an exception because, when assembling the parcel for the customer, the supplies do not correspond to the order list. The Warehouse local diagnoser, from information on data dependencies, can automatically derive that the reason can either be that the supplies sent by one Supplier were wrong, or that the item named in the order list from the Shop were the incorrect ones.

The Global diagnoser, upon receiving this information from the Warehouse local diagnoser, asks to the Supplier local diagnoser to evaluate the hypothesis that it sent the wrong items. Based on the Supplier model, this local diagnoser computes that this can be happened for three reasons: a) when sending the package with the Supplies to the warehouse the wrong items were inserted in it; b) when receiving the order, the wrong codes were written down; c) the initial item codes were wrong. It also computes that in case a) the codes on the shipping note sent to the Warehouse should be correct and should not match the supply contents, while in case b) and c) the codes on the shipping note are also incorrect, and the shipping note should match the contents of the supply.

At this point, the Global diagnoser invokes the Shop local diagnoser in order to evaluate the hypothesis that the Shop may be the source of the problem. The Shop local diagnoser computes that the source may indeed be in the Shop, and in particular that the Shop might have made an error in matching the ordered items to their codes (possibly because of a database error). The Shop local diagnoser also computes that, however, the item names cannot be wrong, because they were provided by the user.

Eventually, the Shop local diagnoser computes that if it made this error, then the item codes in the order list should be wrong. Given these results, the Supervisor invokes again the Warehouse local diagnoser for analyzing the new information, namely to check the predicted consequences of the different fault hypothesis. The Warehouse thus checks the order list against the shipping notes and the following situations can happen: a) The codes in the order list match those in the shipping note, and the shipping note matches the contents of the supply. This means that the responsible for the problem is the Shop. b) The codes in the order list match those on the shipping note, but the shipping note does not match the contents of the supply. Then the responsible for the problem is the Supplier, and the problem occurred while assembling the supply. c) The codes in the order list do not match those on the shipping note. In this case the responsible for the problem is the Supplier, and the problem occurred while writing down the codes at the initial stages.

Repair planning

A composition of Web Services can be considered as a workflow of activities. An invocation of a web service is an execution of an activity which has some pre-defined input and output objects. We also say that output objects are the affected objects of an activity. The input of the repair process is a set of activities and objects which are considered as faulty. All other activities are considered to be correct. However, some correct activities might have produced wrong outputs (i.e. suspect objects) because of wrong inputs or because their invocation was mistakenly caused by a wrong branching of the workflow. Repairing this composition of faulty activities and suspect objects is the task of repair.

Let us assume that in the FoodShop example described above the “split orders” activity of the Shop is faulty (i.e. the single fault diagnosis submitted to the repair process). As mentioned above the wrong product codes for some product names were determined which was discovered when assembling the parcel. Since this activity is performed by a human, there is hope that this failure is intermittent. Consequently, the repair of this faulty activity is just a REDO. Basically all subsequent activities of the “split orders” (Shop) activity in the Shop, Suppliers, and Warehouse must be redone. However, it is not clear which one. This indeterminism depends on the unknown branching of the choosing blocks and on the possibility of fault masking, i.e. although some input objects are faulty some of the output objects could be correct. E.g. some orders to suppliers could have been correct. This can be determined after the REDO of the “split orders” (Shop) activity and by comparing the newly generated output objects with the old ones. Furthermore repair planning has to take into considerations when a compensation action should be applied. Roughly speaking, compensation actions restore the state of objects to a state which was in effect before then objects were affected by an activity. E.g. compensating the effects of “prepare supplies and ship to the warehouse” (Supplier) implies that goods are returned to the supplier. If a compensation action is applied depends on the plan generation. In our scenario it might be more cost efficient to dump the unnecessary items in the warehouse than to send them back.

Repairing a compositional, conversationally complex Web Service consists of plan generation and execution of (repair) actions until the original intended goals of the web service are achieved. These actions also include DO and REDO of activities of the original workflow. Since we have to deal with indeterminism, the generated repair plan is conditional. Reasoning is based mostly on the analysis of the dependencies between the input and output sets of

activities, their preceding relations and dependencies on goal objects.

We propose the following algorithm (repair procedure):

1. Represent the description of a workflow process in terms and models of the planning system;
2. Generate a repair plan using a planner;
3. If the repair plan does not contain any repair actions - exit;
4. Execute first repair action in the set;
5. Evaluate the new state of the workflow instance;
6. React on the new information received by the workflow;

If old plan can be continued go to Step 3 else go to Step 2.

Note that we are interested in generating optimal and save plans. However, it is also worth to consider unsafe plans (in case safe plans do not exist) if the expected cost of repairing is lower than doing nothing.

There are different planning systems that use such planning languages as A^c, PDDL, K-language and their dialects (Eiter et al 03., Huy Tu et al., 04). In most cases knowledge is separated into two groups. First, background knowledge comprises static information that cannot be changed during planning. In our case this background knowledge includes the set of goal objects, input and output parameters of activities, preceding relations between activities. The second part includes the planning code with fluents, actions, and causal/execution rules. This part must be independent of a description of a specific workflow. It includes a general specification applicable to any workflow that is correctly described in the background knowledge.

Depending on the problem there may be many repair plans with different structures. As we see from the algorithm, the most important information that can be obtained from the repair plan is which repair action must be applied first. The rest of the repair plan can be changed or removed, so, actually, only the decision which of the repair actions must be applied first is important.

The evaluation of the new state is needed to decide which changes to the repair plan must be applied. This evaluation must answer the following questions after executing a repair action:

- Was the applied repair action successfully or not?
- What are the new values of the objects in the workflow state?
- What was the exit status of the activity after repair (is there any exit code that shows a fault again)?
- Is there any new information about activities which are long-running transactions?

The analysis after action execution must also consider the results produced and compare them with the states of the objects of the original faulty workflow. Since we do not know the internal behaviour of an activity, its effects may not depend on all its input data. Even if all input data was faulty, it may happen that the effects are correct.

Such analysis can reduce the amount of suspect objects and lead to the need for re-planning since more efficient plans are possible. Note that compared to the original workflow the order of executing activities may change because some activities already produced correct results or because we can easily imagine cases where computing the correct values for choosing blocks is necessary and preferred in order to find out which activities are important to produce correct goal objects. Also, we have to consider new data, obtained from long-time running activities. Completed long-running activity that used suspect objects must be compensated, if needed.

The proposed solution is intended to resolve situations in the fully orchestrated workflows. Moving towards choreography, it is proposed to use the same algorithm for repair plan generation for each workflow in the composition but add additional communication protocols for information exchange.

As depicted in Figure 3 the Global Recovery Planner interacts with the Local Recovery Execution of all services. The task of the Global Recovery planner is to mediate between the Local Recovery Execution services in order to optimize the overall repair activities. Repair plans for each of the local workflows (e.g. Shop, Supplier, and Warehouse) can be generated in a separated session of repair planning. The reasoning process in each of these sessions is performed locally using the same techniques and algorithms. The only difference to the orchestration case is that there must be a communication protocol between sessions. As soon as activities in different workflows share objects as well as invoke and wait for results of each other, this information must be shared also between sessions of repair planning. Note that we not assume that sessions know about the workflow structures of each other. The exchanged information includes:

- the objects which are shared between workflows;
- the states of these input and output objects regarding the correctness;
- the activities which are activated in the partner workflow;
- the existence of local goal objects of the partner workflow and the inputs needed by the partner workflow to achieve these local goals.

By exploiting this information we can propagate information about goals, their needed input objects, and the states of objects from one workflow to another.

In our example, the repair plans for Shop, Supplier and Warehouse workflows will be generated in separated sessions. The Warehouse informs the Suppliers about a local goal, i.e. “deliver the correct parcel to the customer” which depends on correct inputs from the suppliers, e.g. correct items. The correctness of these items depends on the correctness of the input provided by the “split orders” (Shop) activity. Therefore, the Shop workflow has to REDO this “split orders” (Shop) activity. Consequently, goal dependencies are propagated through the composite workflow. Conversely, the communication process has to inform the separated repair planning sessions about the correctness of objects.

Discussion and Future work

The sections above presented the results achieved in the first phase of the WS-DIAMOND project. In this phase we concentrated on some specific problems, making assumptions in order to constrain the problem. Such assumptions are being progressively relaxed or removed in the second part of the project; this approach is allowing us to manage the complexity of the problem and of the task. In the following we analyse the assumptions we made, the way we plan to remove or relax them and the way this will impact the architecture presented in the previous sections.

A first major assumption we made is that we are considering orchestrated services. We are currently extending the approach to deal with choreographed services. This actually does not impact the overall diagnostic architecture (which is not influenced by this distinction except that the global diagnoser must be associated with the owner of the complex process rather than with the orchestrator). On the other hand repair and the self healing layer are being modified.

In the current approach we are also assuming that all services are WS-DIAMOND enables, that is that they have an associated diagnostic service. Such an assumption is being removed and we are considering also the case where some services are black boxes.

Another assumption in the first phase is that we are concentrating on functional errors, while in the second phase we are considering also problems related to the Quality of Service. We will also extend the range of functional faults we are considering. This means, in particular, that the self-healing layer discussed in the previous section is being modified to include also modules for monitoring quality of service parameters.

As regards repair, we worked with a limited set of repair primitives and we are currently extending this set to include further alternatives to be considered during repair planning. On the other hand, in the project we do not expect to remove the general assumption that diagnosis and repair are performed sequentially. The issue of interleaving repair/recovery with (Fridrich, 93) which is a very important one in diagnostic problem solving, will be a topic for future investigations outside the project.

Finally in this phase we assumed that the model of service activities which is needed by its local diagnoser is hand made. However, the dependencies that are needed can be derived from the service description and indeed we are currently investigating how the model can be produced a partially automated way. Moreover we explored how semantic annotations on service properties (e.g., properties of exchanged information) which can be useful as observations to the diagnostic process can be learned from service logs.

Finally, in the second phase of the project we will develop a framework for the analysis of diagnosability and repairability of complex services, starting from previous work and experience in the diagnosticability analysis for artefacts.

References

AI Magazine, Special Issue on Model-based Diagnosis, AAAI, Winter 2003.

ISTAG 2004: Information Society Technologies Advisory Group WG: Grand Challenges in the Evolution of the Information Society, Report (July 2004), ftp://ftp.cordis.lu/pub/ist/docs /istag_draft_report_grand_challenges_wahlster_06_07_04.pdf.

V. Brusoni, L. Console, D. Theseider Duprè, P. Terenziani: A Spectrum of definitions for temporal model-based diagnosis; in *Artificial Intelligence*, Vol 102, no 1, June 1998, pp 39-79.

L. Console, O. Dressler: Model-based diagnosis in the real world: lessons learned and challenges remaining, in *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence. IJCAI 99*, Stockholm, Sweden,1999, pp.1393-1400.

L. Console, C. Picardi, D. Theseider Dupre'. A Framework for Decentralized Qualitative Model-Based Diagnosis. In: *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence. IJCAI-07*. Hyderabad, India, 2007. pp. 286-291. (Preliminary paper in *Proc. 17th Int Work.p on Principles of Diagnosis DX'06*, Penaranda de Duero, Spain, 2006.

T.Eiter, W. Faber, N.Leone: A logic programming approach to knowledge-state planning, II: The DlvK system. *Artificial Intelligence, vol.144,* p.157-211, 2003.

G. Friedrich. Model-based diagnosis and repair. *AI Communications,* Vol. 6(3/4), Sept./Dec. 1993.

I. Grosclaude. Model-based monitoring of component-based software systems, *Proc. 15th Int. Work. on Principles of Diagnosis DX'04*, Carcassonne, France, pp. 155-160, 2004.

W. Hamscher, L. Console, J. de Kleer, *Readings in Model-Based Diagnosis*, Morgan Kaufmann, 1992.

P. Huy Tu, T. Cao Son: Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Casual Laws using Answer Set Programming. *Logic Programming and Nonmonotonic Reasoning,* Springer, pp. 261-274, 2004.

W. Mayer, M. Stumptner, F. Wotawa. Model-Based Debugging or How to Diagnose Programs Automatically, *Proc. IEA/AIE 2002*, Cairns, Australia, Springer LNAI, pp. 746-757, 2002

W. Mayer, M. Stumptner. Debugging Failures in Web Services Coordination, in *Proc. 17th Int. Work. on Principles of Diagnosis DX'06*, Penaranda de Duero, Spain, pp. 171-178, 2006.

B. Peischl, J. Weber, F. Wotava: Runtime fault detection and localization in component-oriented software systems, in *Proc. 17th Int Work. on Principles of Diagnosis DX'06*, Penaranda de Duero, Spain, pp. 195-203, 2006.

Peltz C., Web Service Orchestration and Choreography, *IEEE Computer*, vol. 36, no. 10, 46-52, October 2003.

R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, Vol. 32 (1), 1987, pp. 57-95.

M. Stumptner, F. Wotawa: Modeling Java Programs for Diagnosis, In *Proc. ECAI 2000*, pp. 171-175.

Self-healablity = diagnosability + repairability

Marie-Odile Cordier (+), Yannick Pencolé (*), Louise Travé-Massuyès (*), Thierry Vidal (+)
(+) IRISA/INRIA/University of Rennes1 - Campus de Beaulieu, 35042 Rennes Cédex – FRANCE
email: *firstname.lastname*@irisa.fr
(*) LAAS-CNRS, University of Toulouse - Avenue Colonel Roche, 31077 Toulouse Cédex – FRANCE
email: *firstname.lastname*@laas.fr

Abstract

Real-life complex systems are often required to have a high level of autonomy, even in faulty situations. The diagnosability analysis is the a priori study of the capability of a system to be self-aware about its current state by analysing the observations received by the sensors. The repairability analysis is the a priori study of the capability of a system to react to faults by applying repair actions. Even though they are strongly related, these properties are generally analysed independently. This paper builds upon the state of the art in both domains and proposes a joint property, called self-healability, which achieves a bridge between diagnosability and repairability. We first revisit and extend the classical definitions of diagnosability and repairability. Then, we give our definition of self-healability illustrated with several examples.

1 Introduction

Real-life complex systems are often required to have a high level of autonomy, even in faulty situations. They are expected to be self-aware of their current state and reactive to faults by applying repair actions that take them back to nominal operation. In other words, such systems must be self-healing (Ghosh *et al*. 2007). Designing self-healing systems requires to be able to evaluate the joint degree of self-awareness and reactiveness. In the scientific community, these two properties are better known as diagnosability, i.e. the capability of a system and its monitors to exhibit different observables for different anticipated faulty situations, and repairability, i.e. the ability of a system and its repair actions to cope with any unexpected situation. However these two properties are generally analysed independently. On the one hand, the diagnosis community has proposed a formal definition of diagnosability (Sampath *et al*. 1995; Cordier, Travé-Massuyès, & Pucel 2006) and developped several approaches to check it (Jiang *et al*. 2001; Yoo & Lafortune 2002; Cimatti, Pecheur, & Cavada 2003; Jéron *et al*. 2006; Travé-Massuyès, Escobet, & Olive 2006; Schumann & Pencolé 2007). On the other hand, repairability has received much less attention, and refers to the software systems community that is more interested in control-based approaches such as fault tolerance or dependability as well as to the planning community that uses repair plans often without strong guarantees on their applicability.

A first naive approach could be suggested: first identify the set of all possible basic faults, then analyse separately their diagnosability (i.e. can the observed traces of any fault always be matched back to that fault with certainty ?) and their repairability (i.e. does there exist, for any fault, at least one plan of actions capable of repairing it once it has occurred ?). If one gets positive answers to both questions, then the system is self-healing. But that would be too strong a requirement because, as regards self-healability, complete diagnosability may not be needed: consider for instance that observations can always provide the information that fault f_1 or fault f_2 has occurred, but the system is not always able to discriminate them. One would conclude the system is not diagnosable with respect to these faults. But, if one has a repair plan r_k that is fine for repairing both f_1 and f_2, then this is enough to ensure self-healability! This example calls for a more general and integrated definition of diagnosability and repairability to serve the purpose of self-healability. The basic idea that is developped throughout this paper is to identify a set of so-called *macrofaults* (e.g. 'f_1 or f_2') such that these macrofaults are both diagnosable and repairable.

What our work shows is that facing diagnosability and repairability together calls for extending the classical diagnosability concept. This extension defines diagnosability for macrofaults that may overlap in terms of their underlying sets of basic faults, unlike diagnosability that is classically defined for partitions of basic faults. Self-healability is then defined as achieving a bridge between diagnosability and repairability. The strong form of self-healability checks that there exists at least one set of macrofaults which can always be diagnosed with certainty and be repaired. It is then shown that weakened forms of self-healability can be defined. The first weakened property requires full repairability of the macrofaults but accepts that they are diagnosed with certainty only for a subset of the possible behaviours of the system. The second property accepts partial repairability of a subset of macrofaults but requires that they are diagnosed with certainty for all the possible behaviours of the system.

Section 2 starts with some related work that helps positioning our work within the overall area of dynamic systems supervision, then section 3 gives some preliminary definitions and hypotheses on the faults, the observables and the repair actions. Section 4 focuses on the strong form of self-healability. We first give our definition of diagnosability, which is an extention of the classical definition. We then

define repairability and conclude by joining these two properties to define self-healability. Section 5 defines two weakened forms of self-healability, namely weak self-healability and partial self-healability. Section 6 concludes by giving some perspectives on the way self-healability can be extended to include temporal constraints. Another issue is to show the part that (weakened) self-healability can play at design time to help improving the system properties by augmenting either the observable events or the set of repair actions in a coordinated way.

2 Related Work

In (Ghosh *et al*. 2007) self-healing systems are presented as a rather new and rapidly increasing domain of interest in the software systems community. The definition keeps relatively general, and is stated as:

> *Self-healability is the property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normality.*

That definition is related to the even more general definition of *dependable* systems, that defines the system as globally trustworthy with respect to its ability to always deliver its service, or *fault-tolerant* systems, in which faults that occur do not affect the efficiency of the system. But there are many ways to guarantee such a general property. *Controllability* of dynamic systems (Ramadge & Wonham 1989) is a way to proactively modify the architecture of a system at design time to hinder any occurrence of a catastrophic fault. Another way is to achieve fault-tolerance by on-line mechanisms, to be used whenever a fault occurs. That might be addressed through so-called *passive approaches* that rely on robust feedback mechanisms accounting for possible deviations from the normal behaviour. See for instance (Tornil, Escobet, & Travé-Massuyès 2003)(Garcia, Bernussou, & Arzelier 1994) in the continuous processes domain. In the software domain, approaches like *homeostasis* rely on the same idea, arguing that the distinction between health and unhealth is sometimes tenuous and proposing to maintain a stable internal state for the system despite external variations (Shaw 2002).

On the contrary, self-healing systems are clearly *recovery*-oriented: a clear difference is made between healthy and unhealthy situations, and the system is monitored to

- detect when the system moves from a normal mode to a faulty mode,
- diagnose the situation,
- choose and execute an appropriate recovery strategy.

That principle can be seen as an *active approach* in the fault-tolerant systems community. It may however result, as can be seen in (Saboori & Zad 2005), in integrating recovery modes in the global model of the system, making then tenious the boundary between passive and active approaches.

Our work is clearly positioned towards self-healability, adopting a *diagnose*/*repair*-based view. The model specifying the (normal and faulty) behaviours of the system is clearly separated from the repair strategies that are available for bringing back the system from a faulty mode to a normal mode. In the planning community, *plan revision*, *plan adaptation* and *replanning techniques* are variations of the same general idea: in case of a disruption in the current execution of a plan, the nominal plan is halted and some alternative sequence of actions is executed, the latter being either precomputed off line or computed on line. Our work is somehow related to the work in (Washington, Golden, & Bresina 1999), in which so-called *alternate plans* have been computed off line and uploaded on a space system, and the monitoring module tracks deviations and switch to one of such alternate plans whenever needed, to put the system back in a state in which the nominal plan can be restarted. But such planning approaches show two shortcomings :

- full observability is assumed, turning diagnosis rather straightforward, and
- there is no formal analysis of the 'self-healing' property of the system, i.e. there is no guarantee that there always exists a plan execution that successfully repairs the system whatever fault occurs.

Both issues are addressed in this paper.

Removing the assumption of full observability requires a thorough analysis of *diagnosability*. Diagnosability covers a set of properties that have been studied for many years in different fields. In the context of continuous systems, diagnosability is stated in terms of detectability and isolability (Chen & Patton 1994; Basseville 2001). In the DES context, the first definitions have been proposed in (Sampath *et al*. 1995) and several extensions exist for intermittent faults (Contant, Lafortune, & Teneketsis 2004) and for more generic patterns (Jéron *et al*. 2006). A unified diagnosability definition for continuous systems and discrete-event systems is proposed in (Cordier, Travé-Massuyès, & Pucel 2006). Checking diagnosability is computationally complex and several algorithms have been proposed (Yoo & Lafortune 2002; Jiang *et al*. 2001; Cimatti, Pecheur, & Cavada 2003; Schumann & Pencolé 2007). One of the main purpose of this analysis is to provide an assistance for the design of the system, like sensor placements (Travé-Massuyès, Escobet, & Milne 2001) or communication protocol specifications (Pencolé 2005).

On the other hand, assuring, in any situation and at any time, successfull repair, calls for the definition of a formal *repairability* property, which to our knowledge has never been thoroughly defined.

Ultimately, our work aims at combining diagnosability and repairability to reach a self-healability property, consistent with (Ghosh *et al*. 2007), that can be ensured at design time.

3 Preliminaries

In this paper, we adopt the generic viewpoint defined in (Cordier, Travé-Massuyès, & Pucel 2006). A system may be a discrete-event system or a continuous system. Depending on its inputs, the system may have several *behaviours* and produce several outputs. We first define preliminary notions that are used throughout this paper.

3.1 Faults

The set of possible faults $\mathcal{F}$ of the system is composed of nf *basic faults* $\mathcal{F} = \{f_1, \ldots, f_{nf}\}$. This paper relies on the following single fault assumption.

Assumption 1 *If the behaviour of the system is faulty, only one fault is present.*

In the following and without loss of generality, f_i denotes either the type of the fault or its presence in the system. Moreover, in order to be generic, we extend $\mathcal{F}$ with the symbol $norm$ which stands for the absence of fault. In the following, $norm$ is considered just like any other f_i of $\mathcal{F}$.

Definition 1 (Macrofault) *A* macrofault F_j *is a set of faults* $F_j \subseteq \mathcal{F}, F_j \neq \emptyset$. *If* F_j *is present in the system, it means that one and only one of the basic faults* $f_i \in F_j$ *is present in the system.*

For instance, the macrofault $\{f_1, f_2\}$ represents the fact that either f_1 or f_2 is present, the macrofault $\{f_2, norm\}$ means that either the fault f_2 is present or there is no fault.

A macrofault may be a singleton ($F_j = \{f_i\}$), i.e. basic faults may be seen as special cases of macrofaults. A set of macrofaults is then denoted by $E(\mathcal{F})$ (with $E(\mathcal{F}) \subseteq 2^{\mathcal{F}}$).

Definition 2 (Covering set) *A set of macrofaults* $E(\mathcal{F})$ *is a* covering set *of* $\mathcal{F}$ *if and only if* $\forall f_i \in \mathcal{F}, \exists F_j \in E(\mathcal{F})$ *such that* $f_i \in F_j$.

3.2 Observations

Depending on the type of system, observations consist of sequences of observable events or sets of measured values for observable variables. These observations are the observable part of the system's behaviour. This paper focuses on defining the generic notion of self-healability which does not require the distinction between these different types of observations. Thus, for the sake of generality, the observability of the system is represented by the set OBS of all the possible observations of the system. $\sigma \in OBS$ is called an *observable* of the system.

In the case of discrete event systems (DES), OBS is usually specified through the set of observable events $\mathcal{O} = \{o_1, ..., o_{no}\}$. Complementing $\mathcal{O}$ with the set of unobservable events $\mathcal{U} = \{u_1, ..., u_{nu}\}$ determines the whole set of events of the system $\mathcal{E} = \mathcal{O} \cup \mathcal{U}$. Faults are specific unobservable events $\mathcal{F} \subseteq \mathcal{U}$.

In the case of continuous systems, OBS is usually specified as the set of all possible observation tuples for observable variables, i.e., $OBS = \{(o_1, o_2, ..., o_k)\}$ where k is the number of sensoring devices. Faults can be thought as unknown disturbances that affect the system's behaviour and hence the observations.

3.3 Repairs

We first define, in a simplified way, what a repair plan is. Planning contexts may of course require more elaborated definitions, but the global property definitions proposed in this article remain the same. For our purpose, we do not really need to know what a repair plan is semantically speaking, we merely need to have a set of such repair plans and to be able to match them to (basic) faults.

Definition 3 (Repair Plan) *A* repair plan r_k *is defined as a sequence of actions* $A_k = \{a_{k1}, ..., a_{kn}\}$, *each* a_{ki} *belonging to the set of all possible recovering basic actions (which are seen as events in DES). A repair plan is always assumed to be applicable (no preconditions). Each repair plan* r_k *has a goal* g_k *which is the nominal state of the system in which the repair plan is expected to bring the system back.*

The set of available repair plans is denoted $\mathcal{R} = \{r_1, ..., r_{nr}\}$ One can see that the definition of a repair plan is independent from the faults it might apply to, one then needs to establish that relation (just like signatures are sets of observations that are related to faults). That is done through the predicate $Repair$:

$Repair(r_k, f_i)$ means that the repair plan r_k, if executed in a state in which f_i is present, brings the system back to a state in which the goal g_k should be true and $norm$ present[1].

The $Repair$ predicate can also be applied to a macrofault, and there is an equivalence between repairing a macrofault and repairing each of the basic faults belonging to the macrofault, i.e.

Property 1 $Repair(r_k, F_j) \equiv \forall f_i \in F_j, Repair(r_k, f_i)$.

4 Self-healability

Self-healability is intuitively defined by the following:

A system is self-healing if, and only if, after the occurence of any basic fault, a diagnosis is issued that automatically raises a repair plan fitted for the fault.

Behind this intuitive definition, two properties of the system are hidden: diagnosability and repairability. What is also hidden is *what* the system should be able to diagnose and *which* plan should then be applied. In this section, we first introduce these two notions to finally propose a first definition for self-healability.

4.1 Diagnosability

A first requirement for a diagnosable system is that every behaviour (faulty or not) must produce some observations. If no observation is available, the system is not diagnosable at all. Therefore, one requirement for self-healability is:

Property 2 *The system is not silent: every possible behaviour has an associated observable.*

In the case where the observations are the result of some measurements (in a continuous system for instance), this property is usually satisfied since the measures can be performed whatever the behaviour of the system really is. In the case of discrete-event models, this property implies some consequences on the system itself because the observations are usually events emitted by the system (in the literature about DES, this property is usually called the *liveness* of the observations (Sampath *et al.* 1995)).

Diagnosability analysis relies on the notion of *fault signatures* (Cordier, Travé-Massuyès, & Pucel 2006). Intuitively,

[1] We consider that there always exists a repair plan r_{norm} such that $Repair(r_{norm}, norm)$, r_{norm} being the empty plan with a set of actions $A_k = \emptyset$.

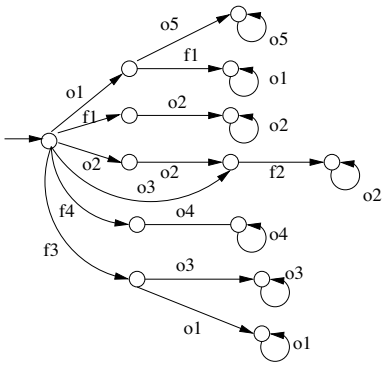


Figure 1: An example.

a fault signature is the association between a fault and a set of possible observations. The following definition defines the concept of elementary signature, noted *e-signature*. An e-signature corresponds to one observable in the presence of the fault and in a given context.

Definition 4 (Fault e-signature) *An* e-signature σ *of* F_j *is an observable such that there exists a behaviour of the system for which* σ *is observed when a fault* $f_i \in F_j$ *is present.*

Example:

Figure 1 presents the global model of a discrete-event system Σ. The set of basic faults is $\mathcal{F} = \{norm, f_1, f_2, f_3, f_4\}$ and the fault events are not observable. The other events (i.e. o_1, o_2, o_3, o_4 and o_5) are observable.

$o1o5^\infty$ is an e-signature of the normal behaviour; it is composed of one observation $o1$ followed by an infinite sequence of $o5$. $o2^\infty$ is an e-signature of the fault $\{f_1\}$ and of the fault $\{f_2\}$, therefore it is also an e-signature of $\{f_1, f_2\}$, but it can also be considered as an e-signature of, for instance, $\{f_1, f_2, f_3\}$.

The observation of an e-signature σ denotes that the fault F_j may be present in the system. The presence of F_j is not certain since it is possible that σ is also the e-signature of another fault $F_{j'}$.

Definition 5 (Characteristic e-signature) *An e-signature* σ characterizes F_j *if and only if in every behaviour that produces the e-signature* σ*, a fault* $f_i \in F_j$ *is present.*

If an e-signature σ *characterizes* the presence of F_j, it means that the observation of σ guarantees that the fault F_j is certainly present in the system. Note that an e-signature that characterizes F_j may characterise a fault $F_{j'}$ where $F_{j'} \cap F_j \neq \emptyset$, and any e-signature characterizes the macrofault $\mathcal{F}$.

In other words, when observing a characteristic σ, one is sure that one of the $f_i \in F_j$ has occurred, and that other $f_{i'}$ which are not in F_j have not occurred. But each $f_i \in F_j$ may have other e-signatures that will be characteristic of other macrofaults that also include f_i.

Example : $o3o2^\infty$ is a characteristic e-signature of the fault $\{f_2\}$ and of the fault $\{f_1, f_2\}$. $o2^\infty$ is a characteristic e-signature of the fault $\{f_1, f_2\}$.

Definition 6 (Characteristic signature) *A* characteristic signature *of* F_j *is a set of e-signatures that characterise* F_j*. Such a set is denoted* $Sig(F_j)$.

Example : $\{o1^\infty, o2^\infty\}$, $\{o2^\infty, o3o2^\infty\}$,$\{o1^\infty, o3^\infty\}$ are the respective sets of all the e-signatures of $\{f_1\}$,$\{f_2\}$, $\{f_3\}$ but are not characteristic signatures of $\{f_1\}$,$\{f_2\}$, $\{f_3\}$. $\{o4^\infty\}$ is the set of all the e-signatures of $\{f_4\}$ and is a characteristic signature of $\{f_4\}$, noted $Sig(\{f_4\})$. $\{o1^\infty, o3^\infty\}$ is a characteristic signature of $\{f_1, f_3\}$, as well as $\{o3^\infty\}$ and $\{o1^\infty\}$.

Now, we are ready to propose a definition for diagnosability that is suitable for self-healability. Classically, diagnosability is formally defined on a partition of the basic faults of $\mathcal{F}$ (Cordier, Travé-Massuyès, & Pucel 2006). In this paper, the definition of diagnosability is extended to a covering set of faults $E(\mathcal{F}) = \{F_1, \ldots, F_m\}$.

Definition 7 (Diagnosability of a set of macrofaults) *The covering set* $E(\mathcal{F})$ *is* diagnosable *if and only if there exists* m *sets of characteristic signatures* $Sig(F_1), \ldots, Sig(F_m)$ *such that:*

1. $\bigcup_{i=1}^{m} Sig(F_i) = OBS;$
2. $\forall i, j, i \neq j, Sig(F_i) \cap Sig(F_j) = \emptyset.$

In other words, the set $PSig = \{Sig(F_1), \ldots, Sig(F_m)\}$ is a *partition* of the e-signatures, and $E(\mathcal{F})$ is *diagnosable* if and only if a partition $PSig$ exists such that the observation of $\sigma \in Sig(F_j)$ guarantees that F_j is present.

Example: The set of macrofaults $E_0(\mathcal{F}) = \{\{norm, f_1, f_2, f_3, f_4\}\}$ is diagnosable, the partition of e-signatures is $PSig_0 = \{\{o1o5^\infty, o_1^\infty, o_2^\infty, o_3o_2^\infty, o_3^\infty, o_4^\infty\}\}$. The set of macrofaults $E_1(\mathcal{F}) = \{\{norm\}, \{f_1, f_2, f_3\}, \{f_4\}\}$ is diagnosable, the partition of e-signatures is $PSig_1 = \{\{o1o5^\infty\}, \{o_1^\infty, o_2^\infty, o_3o_2^\infty, o_3^\infty\}, \{o_4^\infty\}\}$. The set $E_2(\mathcal{F}) = \{\{norm\}, \{f_1, f_2\}, \{f_1, f_3\}, \{f_4\}\}$ is also diagnosable and the corresponding partition is $PSig_2 = \{\{o1o5^\infty\}, \{o_2^\infty, o_3o_2^\infty\}, \{o_1^\infty, o_3^\infty\}, \{o_4^\infty\}\}$. The set $E_3(\mathcal{F}) = \{\{norm\}, \{f_1\}, \{f_2\}, \{f_3\}, \{f_4\}\}$ is not diagnosable because there are some cases in which f_1 and f_2 are not discriminated (the same for f_1 and f_3) . $E_4(\mathcal{F}) = \{\{norm\}, \{f_1\}, \{f_2, f_3\}, \{f_4\}\}$ is not diagnosable. $E_5(\mathcal{F}) = \{\{norm\}, \{f_1, f_3\}, \{f_2\}\{f_4\}\}$ is not diagnosable.

This diagnosability definition is generic and covers different concepts. For instance, if there is a macrofault in $E(\mathcal{F})$ which contains $norm$ and at least a basic fault f_i, and $E(\mathcal{F})$ is diagnosable, it means the system is not *fault-detectable* with respect to that set of macrofaults: some faulty behaviour, whatever the fault is, may not be detected. On the contrary, if $\{norm\} \in E(\mathcal{F})$ and $E(\mathcal{F})$ is diagnosable and none of the other macrofaults contain $norm$ then all faulty behaviours are detectable.

In the specific case where $E(\mathcal{F})$ is a partition of the basic faults with $\{norm\} \in E(\mathcal{F})$, the diagnosability of $E(\mathcal{F})$ corresponds to the classical definition over a partition of basic faults (see (Cordier, Travé-Massuyès, & Pucel 2006)).

We use the predicate $Diagnosable(E(\mathcal{F}))$ to tell that $E(\mathcal{F})$ is diagnosable.

Then the definition of the diagnosability of the system, through the simple predicate $Diagnosable$, comes directly:

Definition 8 (Diagnosability) $Diagnosable \equiv \exists E(\mathcal{F})$ *such that* $Diagnosable(E(\mathcal{F}))$

Let us remark that a system is always diagnosable as it is always true that $Diagnosable(\{\mathcal{F}\})$.

4.2 Repairability

Repairability is actually straightforward and comes directly from the basic definition of sets of macrofaults and through the predicate $Repair$. First, a macrofault F_j is repairable if and only if there exists a repair plan that repairs it:

Definition 9 (Repairability of a macrofault) $Repairable(F_j) \equiv \exists r_k$ *such that* $Repair(r_k, F_j)$

Then, considering the basic faults belonging to a macrofault, we have the implication:

Property 3 $Repairable(\{f_i, f_j\}) \models Repairable(f_i)$ *and* $Repairable(f_j)$

The converse is of course not always true (only when a common plan repairs both f_i and f_j).

The repairability of a set of macrofaults is then defined as the repairability of all the macrofaults in the set:

Definition 10 (Repairability of a set of macrofaults) $Repairable(E(\mathcal{F})) \equiv \forall F_j \in E(\mathcal{F}) Repairable(F_j)$

The repairability of a system (that we simply note through the predicate $Repairable$ with no parameters) means that there exists a set of macrofaults which covers all the basic faults and that is repairable.

Definition 11 (Repairability of the system) $Repairable \equiv \exists E(\mathcal{F})$ *such that* $Repairable(E(\mathcal{F}))$ *and* $E(\mathcal{F})$ *is a covering set*

Example : If the only repair plan is r, with $Repair(r, \{f_1, f_3\})$, we get Repairable($\{f_1$,$f_3\}$), and also Repairable($\{f_1\}$) and Repairable($\{f_3\}$). However, the system is clearly not repairable as the faults f_2 and f_4 are not repairable.

4.3 Self-healability

Our first definition for self-healibility directly derives from the definitions of diagnosability and repairability.

Definition 12 (Self-healing covering set) *A covering set* $E(\mathcal{F})$ *is* self-healing *iff it is diagnosable and repairable, i.e.* $SelfHealing(E(\mathcal{F})) \equiv Diagnosable(E(\mathcal{F}))$ *and* $Repairable(E(\mathcal{F}))$

Given a self-healing covering set $E(\mathcal{F})$, the observation of a signature $\sigma \in OBS$ is always associated to a macrofault $F_\sigma \in E(\mathcal{F})$ because $E(\mathcal{F})$ is diagnosable. Moreover, since $E(\mathcal{F})$ is repairable, it means that there exists a repair plan for F_σ (a repair plan that repairs any basic fault of F_σ), therefore F_σ can be repaired. Finally, $E(\mathcal{F})$ is a covering set so every basic fault is covered by at least one macrofault, so every basic fault can be healed.

Definition 13 (Self-healing system) *A system is* self-healing *iff there exists a self-healing covering set* $E(\mathcal{F})$*, i.e.* $SelfHealing \equiv \exists E(\mathcal{F})$ *such that* $E(\mathcal{F})$ *is a covering set and* $SelfHealing(E(\mathcal{F}))$.

Example :

- Case 0 : Repairable($norm$) and Repairable($\{f_1$,$f_3\}$) and Repairable($\{f_1$,$f_2\}$) and Repairable(f_4)

  The set $E_2(\mathcal{F}) = \{\{norm\}, \{f_1, f_2\}, \{f_1, f_3\}, \{f_4\}\}$ is diagnosable and repairable. The system is then self-healing. It would also have been the case if we had repair plans such that Repairable($\{f_1$,f_2,$f_3\}$) and Repairable(f_4) because $E_1(\mathcal{F}) = \{\{norm\}, \{f_1, f_2, f_3\}, \{f_4\}\}$ is diagnosable.

- Case 1 : Repairable($norm$) and Repairable($\{f_1$,$f_3\}$) and Repairable(f_2) and Repairable(f_4)

  No set of macrofaults is self-healing as there does not exist a repair plan for $\{f_1, f_2\}$. The system is thus not self-healing.

As said previously, in a self-healing system every basic fault can be repaired (even in cases where it cannot be fully distinguished from other faults). This property is the strong form of self-healability. In the next section, we propose weakened forms of self-healability that can also be of interest, for instance during the design step.

5 Weakened forms of self-healability

The strong form of self-healability seen above is exactly what is needed when checking a system as it indicates that each basic fault can always be repaired, either because it can always be diagnosed and repaired, or because a macrofault containing it can always be diagnosed and repaired. In other words, it means that, for any occurrence of a fault, the observations and repair plans are sufficient to ensure a sufficiently precise diagnosis enabling to trigger a repair plan.

However, weakened forms of self-healability are interesting. For instance, when designing a system, it can be useful to guide the designer to improve the current self-healability. It can also be useful to identify a subpart of basic faults which are self-healing even if the whole system is not.

In the following, we define two weakened forms of self-healability. The first one, called *weak self-healability*, consists in assuring that each fault can be diagnosed and repaired at least in some contexts. The second one, called *partial self-healability*, consists in assuring that there exists a subset of faults that are always self-healing.

5.1 Weak Self-Healability

A basic fault can manifest itself according to a set of e-signatures. Weak self-healability is a way to decide if, at least for a subset of its e-signatures, each basic fault is repairable.

Weak Self-Healability relies on weak diagnosability and repairability. Let us first define what we call weak diagnosability.

Weak Diagnosability A covering set of macrofaults $E(\mathcal{F})$ is weakly diagnosable iff for each macrofault of $E(\mathcal{F})$, there exists at least an e-signature which characterises it. However, it may also exist e-signatures which do not characterize any macrofault of $E(\mathcal{F})$.

Definition 14 (Weak Diagnosability of a set of macrofaults) *The set of n macrofaults $E(\mathcal{F})$ is* weakly diagnosable *iff it is a covering set and there exists a partition of e-signatures $PSig = \{Sig_1, \ldots, Sig_r\}$ such that, $r \geq n$, for all $i \in \{1, \ldots, r\}, Sig_i \neq \emptyset$ and for all $i \in \{1, \ldots, n\}$, the signature Sig_i characterizes the macrofault F_i.*

Another way of expressing weak diagnosability is to say that the covering set of macrofaults $E(\mathcal{F})$ is *weakly diagnosable* iff there exists a set of macrofaults $E'(\mathcal{F})$, such that $E'(\mathcal{F})$ is diagnosable and $E(\mathcal{F}) \subseteq E'(\mathcal{F})$.

We will note weak diagnosability of a set of macrofaults $WeaklyDiagnosable(E(\mathcal{F}))$ and consequently weak diagnosability of the whole system can be defined as

Definition 15 (Weak Diagnosability) $WeaklyDiagnosable \equiv \exists E(\mathcal{F})$ *such that* $WeaklyDiagnosable(E(\mathcal{F}))$

As seen before, a system is always diagnosable and thus it is also always weakly diagnosable.

Example:

By definition, every diagnosable $E(\mathcal{F})$ is also weakly diagnosable, so $E_0(\mathcal{F})$, $E_1(\mathcal{F})$, $E_2(\mathcal{F})$ are weakly diagnosable. If we consider the set $E_3(\mathcal{F}) = \{\{norm\}, \{f_1\}, \{f_2\}, \{f_3\}, \{f_4\}\}$, it is not weakly diagnosable as $\{f_1\}$ has no characteristic e-signature. In the same way, $E_4(\mathcal{F})$ is not weakly diagnosable. However, $E_5(\mathcal{F})$ is weakly diagnosable with the corresponding partition of e-signatures $PSig_5 = \{\{o_1 o_5^\infty\}, \{o_1^\infty o_3^\infty\}, \{o_3 o_2^\infty\}, \{o_4^\infty\}, \{o_2^\infty\}\}\}$. The first four e-signatures characterize the first four macrofaults; the last one, o_2^∞, is a supplementary one which does not characterize any macrofault of the set $E_5(\mathcal{F})$.

If $DiagnosableSet(\mathcal{F})$ and $WeaklyDiagnosableSet(\mathcal{F})$ are the sets of sets of macrofaults $E(\mathcal{F})$ such that $E(\mathcal{F})$ is diagnosable and weakly diagnosable, respectively. In this example, we have $\{E_0(\mathcal{F}), E_1(\mathcal{F}), E_2(\mathcal{F}), E_5(\mathcal{F})\} \subseteq WeaklyDiagnosableSet(\mathcal{F})$ and by definition, one always gets: $DiagnosableSet(\mathcal{F}) \subseteq WeaklyDiagnosableSet(\mathcal{F})$.

Weak Self-healability The definition of weak self-healability follows:

Definition 16 (Weak self-healability of a set of macrofaults) *The covering set of macrofaults $E(\mathcal{F})$ is* weakly self-healing *iff it is weakly diagnosable and repairable.*

It can be shown from what was said before that the set of macrofaults $E(\mathcal{F})$ is *weakly self-healing* iff there exists a set of macrofaults $E'(\mathcal{F})$, such that $E'(\mathcal{F})$ is self-healing and $E(\mathcal{F}) \subseteq E'(\mathcal{F})$.

Definition 17 (Weak self-healability of the system) *A system is weakly self-healing iff there exists a covering set $E(\mathcal{F})$ that is weakly self-healing.*

Here again we can use a predicate notation:
$WeaklySelfHealing \equiv \exists E(\mathcal{F})$ such that $WeaklySelfHealing(E(\mathcal{F}))$.

Hence the following propositions:

Proposition 1 *1. If a system is self-healing then it is weakly self-healing.*

2. If a system is weakly self-healing, it may not be self-healing.

Proof: The proof of 1 comes from the definitions. The proof of 2 comes from the case 1 of the following example.

Example :

- Case 0 : Repairable($norm$) and Repairable($\{f_1$,$f_3\}$) and Repairable($\{f_1$,$f_2\}$) and Repairable(f_4)
  The set $E_2(\mathcal{F}) = \{\{norm\}, \{f_1, f_2\}, \{f_1, f_3\}, \{f_4\}\}$ is diagnosable and repairable. The system is then self-healing and consequenly also weakly self-healing.
- Case 1 : Repairable($norm$) and Repairable($\{f_1$,$f_3\}$) and Repairable(f_2) and Repairable(f_4)
  No set of macrofaults is self-healing. $E_5(\mathcal{F}) = \{\{norm\}, \{f_1, f_3\}, \{f_2\}\{f_4\}\}$ is not self-healing as it is not diagnosable, but it is weakly self-healing as it is weakly diagnosable and repairable. The system is thus not self-healing but weakly self-healing.
- Case 2 : Repairable($norm$) and Repairable(f_1) and Repairable(f_2) and Repairable(f_3)and Repairable(f_4)
  No set of macrofaults is either self-healing or weakly self-healing as f_1 can never be diagnosed with certainty (even in some specific contexts). The system is thus not self-healing and not weakly self-healing.

5.2 Partial Self-Healability

Another weakened form of self-healability is to guarantee that a subset of macrofaults (or basic faults) are self-healing, even if not all of them. The idea is to guarantee that there exists a subset of basic faults which are self-healing, and thus both diagnosable and repairable.

We first need to introduce the following definition.

Definition 18 (No-overlap property) *The set of sets Y has the no-overlap property with the set of sets X iff $Y \subseteq X$ and $\forall y \in Y \ \forall x \in X \setminus Y \ x \cap y = \emptyset$.*

If $X = \{\{f1\}, \{f1, f2\}, \{f3\}\}$ then $Y = \{\{f1\}, \{f1, f2\}\}$ has the no-overlap property with X but $Y = \{\{f1\}\}$ has not.

Definition 19 (Partial Self-Healability of a set of macrofaults) *The (non covering) set of macrofaults $E(\mathcal{F})$ is* partially self-healing *iff there exists a set of macrofaults $E'(\mathcal{F})$, such that $E'(\mathcal{F})$ is diagnosable and $E(\mathcal{F})$* has the no-overlap property with $E'(\mathcal{F})$ *and $E(\mathcal{F})$ is repairable.*

The no-overlap property is important as it ensures that each macrofault of $E(\mathcal{F})$ can be diagnosed in any context.

Definition 20 (Partial self-healability of the system) *A system is partially self-healing iff there exists a set $E(\mathcal{F})$ that is partially self-healing.*

Here again we can use a predicate notation:
$PartiallySelfHealing \equiv \exists E(\mathcal{F})$ such that $PartiallySelfHealing(E(\mathcal{F}))$.

Let us remark that, as we consider the normal behaviour as a basic fault, denoted $norm$, and if there exists a repair plan, for instance the empty one, associated to $norm$, then a detectable system is always partially self-healing.

Example :

- Case 0 : Repairable($norm$) and Repairable($\{f_1$,$f_3\}$) and Repairable($\{f_1$,$f_2\}$) and Repairable(f_4)
  The set $E_2(\mathcal{F}) = \{\{norm\}, \{f_1, f_2\}, \{f_1, f_3\}, \{f_4\}\}$ is diagnosable and repairable. The system is then self-healing and consequenly also weakly and partially self-healing.
- Case 1 : Repairable($norm$) and Repairable(f_1,f_3) and Repairable(f_2) and Repairable(f_4)
  $E_6(\mathcal{F}) = \{\{f_4\}\}$ is partially self-healing as it exists a covering set of macrofaults $E_2(\mathcal{F})$ such that $E_6(\mathcal{F}) \subset E_2(\mathcal{F})$, $E_6(\mathcal{F})$ and $E_2(\mathcal{F})$ has the no-overlap property, $E_2(\mathcal{F})$ is diagnosable, and $E_6(\mathcal{F}) = \{\{f_4\}\}$ is repairable. It shows that the basic fault f_4 can be repaired for any of its occurrences.
  The set $E_9(\mathcal{F}) = \{\{f_1, f_3\}\}$ is not partially self-healing. It exists a covering set of macrofaults $E_2(\mathcal{F})$ such that $E_9(\mathcal{F}) \subset E_2(\mathcal{F})$, $E_2(\mathcal{F})$ is diagnosable, and $E_9(\mathcal{F}) = \{\{f_1, f_3\}\}$ is repairable but $E_9(\mathcal{F})$ and $E_2(\mathcal{F})$ do not satisfy the no-overlap property; it means that there exists at least an e-signature, here o_2^∞, which is an e-signature of a macrofault of the considered set, here of $\{f_1, f_3\}$, but also of another macrofault not in the considered set, here, $\{f_1, f_2\}$. Even if $E_9(\mathcal{F})$ is repairable, it means that the basic fault f_1 cannot be repaired in any contexts. It would only be true if we had also a repair plan for $\{f_1, f_2\}$.
  The system is thus not self-healing but weakly self-healing and partially self-healing.
- Case 2 : Repairable($norm$) and Repairable(f_1) and Repairable(f_2) and Repairable(f_3) and Repairable(f_4)
  The system is not self-healing and not weakly self-healing. However, as in case 1, $E_6(\mathcal{F}) = \{\{f_4\}\}$ is partially self-healing and the system is thus partially self-healing.
- Case 3 : Repairable($norm$) and Repairable($\{f_1$,$f_2\}$) and Repairable($\{f_1$,$f_3\}$)
  Again, the system is not self-healing and not weakly self-healing. However, $E_7(\mathcal{F}) = \{\{f_1, f_2\}, \{f_1, f_3\}\}$ is partially self-healing as it exists a covering set of macrofaults $E_2(\mathcal{F})$ such that $E_7(\mathcal{F}) \subset E_2(\mathcal{F})$, $E_7(\mathcal{F})$ and $E_2(\mathcal{F})$ has the no-overlap property, $E_2(\mathcal{F})$ is diagnosable, and $E_7(\mathcal{F}) = \{\{f_1, f_2\}, \{f_1, f_3\}\}$ is repairable. The two macrofaults $\{f_1, f_2\}$ and$\{f_1, f_3\}$ can be diagnosed with certainty and are both repairable. It demonstrates that the basic faults f_1, f_2 and f_3 can thus be repaired for any of their occurrences. The system is partially self-healing.
  $E_8(\mathcal{F}) = \{\{f_1, f_2, f_3\}\}$ is not partially self-healing as there does not exist any repair plan for $\{f_1, f_2, f_3\}$. In case we had a repair plan for $\{f_1, f_2, f_3\}$, $E_7(\mathcal{F}) = \{\{f_1, f_2, f_3\}\}$ would be partially self-healing (wrt to $E_1(\mathcal{F})$).

To end up with that property, one could notice that partial self-healability could be related to a partial repairability property. The search for a partially self-healing set of macrofaults will indeed usually be driven by partial repairability: if one knows that some faults are not repairable, it is enough to check diagnosability for the remaining.

Partial repairability means that there exists a set of macrofaults (not necessarily covering all the basic faults) that is repairable.

Definition 21 (Partial repairability of the system) $PartiallyRepairable \equiv \exists E(\mathcal{F})$ *such that* $Repairable(E(\mathcal{F}))$

Property 4 $Repairable \models PartiallyRepairable$

In other words, partial self-healability is consistent (though it is not mandatory) with a system that is only partially repairable. On the other hand, a system that is self-healing or weakly self-healing must necessarily be (fully) repairable.

Property 5 $SelfHealing \models Repairable$

Property 6 $WeaklySelfHealing \models Repairable$

Property 7 $PartiallySelfHealing \models PartiallyRepairable$

To summarize, a self-healing set of macrofaults is such that each macrofault is diagnosable and repairable, which means also, as it is a covering set, that each basic fault is repairable (but not necessarily diagnosable).

A weakly self-healing set of macrofaults is such that, for each macrofault, there exist some contexts, generally not all, in which the macrofaults are diagnosable and repairable, which means also, as it is a covering set, that there are contexts, generally not all, in which each basic fault is repairable.

A partially self-healing set of macrofaults characterises the only subset of basic faults (the ones it covers) which are always repairable (even if they are not necessarily diagnosable).

Other weakened forms of self-healability could be defined and be interesting during the design phase of a system, as for instance coupling weak and partial self-healability, i.e. the sets of macrofaults which would not be but could become weakly self-healing by adding some repair plans.

6 Discussion

The main contribution of this paper is to propose a thorough and integrated definition of the *self-healability* of a dynamic system, through the analysis of what the system needs to recover from faults, when considering together its diagnosis and repair capabilities. Interestingly enough, the system does neither need to provide diagnosability for each basic fault nor to command a set of distinct specialized repair plans, one for each basic fault, but it needs to find a set of 'sufficient' diagnosable sets that can be matched to 'sufficient' repairs. As far as we know, it is the first time that such

a definition is issued. Of course that is only a first step that calls for a lot of extensions.

First, in cases for which self-healability is met, there may exist many different sets of macrofaults establishing the property. An interesting concern is then to determine the 'best' (e.g. the one that provides the most information to the user, or the one that favours repairs that put back the system in preferred states, etc), and the search strategy (and hence the algorithms) to provide that optimal set of macrofaults.

Second, as we have shown, it is not always possible to find a set of macrofaults ensuring self-healability, and one may only get what we have called partial or weak self-healability. These weakened properties are useful at design time because it means that one needs to modify either the observables or the repair plans (i.e. increase sensor or actuator performance) that are available in the system. The interesting issue is then to find the 'best' set of macrofaults to isolate, i.e. here the one that minimizes the needs for extra diagnosis/repair capabilities. As we have shown, we could look for a set that is perfectly diagnosable but not repairable, or perfectly repairable but not always diagnosable, but such sets may be far from being optimal: hence the most interesting set of macrofaults to start with may very well be a set that is neither perfectly diagnosable nor repairable.

Other issues that we would like to deal with in the future are extensions of the current framework to address more elaborated (and more realistic) cases, mainly:

- Multiple faults: if f_2 occurs some time after f_1, maybe it is better to wait and apply a repair plan that will repair both than applying two successive plans...? Then we will need to be more precise in terms of the definition of repair plans and their chracterization in terms of their capabilities to repair only one or several faults, and to be able to repair a fault even though another one has occurred, etc.
- Temporal constraints: a very important issue is to consider the 'delay' that is needed to diagnose a fault, due to waiting for a sufficient number of observations; that delay may counteract some emergency delay for an adequate repair plan, i.e. the delay after which it will be too late to apply the plan. Such delays should be incorporated into our definitions to only match repair plans to macrofaults when such delay constraints are not violated.

References

Basseville, M. 2001. On fault detectability and isolability. *European Journal of Control* 7(8):625–637.

Chen, J., and Patton, R. 1994. A re-examination of fault detectability and isolability in linear dynamic systems. *Proceedings of the 2nd Safeprocess Symposium, Helsinki (Finland)*.

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI'03*.

Contant, O.; Lafortune, S.; and Teneketsis, D. 2004. Diagnosis of intermittent faults. *Journal of Discrete Event Dynamic Systems: Theory and Applications* 14:171–202.

Cordier, M.-O.; Travé-Massuyès, L.; and Pucel, X. 2006. Comparing diagnosability in continous and discrete-event systems. In González, C.; Escobet, T.; and Pulido, B., eds., *17th International Workshop on Principles of Diagnosis*, 55–60.

Garcia, G.; Bernussou, J.; and Arzelier, D. 1994. Robust stabilization of discrete-time linear systems with norm-bounded time-varying uncertainty. *Systems & Control Letters archive, Volume 22 , Issue 5* 327–339.

Ghosh, D.; Sharman, R.; Rao, H. R.; and Upadhyaya, S. 2007. Self-healing systems survey and synthesis. *Decision Support Systems* 42(4):2164–2185.

Jéron, T.; Marchand, H.; Pinchinat, S.; and Cordier, M.-O. 2006. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems, WODES'06*.

Jiang, S.; Huang, Z.; Chandra, V.; and Kumar, R. 2001. A polynomial time algorithm for diagnosability of discrete event systems. *IEEE Transactions on Automatic Control* 46(8):1318–1321.

Pencolé, Y. 2005. Assistance for the design of a diagnosable component-based system. In *17th International Conference on Tools with Artificial Intelligence (ICTAI 05)*, 549–556. IEE Computer Society.

Ramadge, P., and Wonham, W. 1989. The control of discrete event systems. *Proceedings of the IEEE* 81–98.

Saboori, A., and Zad, S. H. 2005. Fault recovery in discrete event systems. In *Proceedings of the ICSC congress on Computational Intelligence Methods and Applications (CIMA'05)*.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete event system. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Schumann, A., and Pencolé, Y. 2007. Scalable diagnosability checking of event-driven system. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI07)*, 575–580.

Shaw, M. 2002. Self-healing: Softening precision to avoid brittleness. In *Proceedings of the First Workshop on Self-Healing Systems WOSS '02, Charleston, South Carolina, USA, November 18-19.*, 111–113. ACM.

Tornil, S.; Escobet, T.; and Travé-Massuyès, L. 2003. Robust fault detection using interval models. In *12th European Control Conference ECC03, Cambridge, UK*.

Travé-Massuyès, L.; Escobet, T.; and Milne, R. 2001. Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI'01*, volume 1, 551–556.

Travé-Massuyès, L.; Escobet, T.; and Olive, X. 2006. Diagnosability analysis based on component supported analytical redundancy relations. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans, Vol. 36, N6*.

Washington, R.; Golden, K.; and Bresina, J. 1999. Plan execution, monitoring, and adaptation for planetary rovers. In *Proceedings of the IJCAI'99 Workshop on 'Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World'*.

Yoo, T., and Lafortune, S. 2002. Polynomial-time verification of diagnosability of partially-observed discrete-event systems. *IEEE Transactions of Automatic Control* 47(9):1491–1495.

A Discrete Event Approach to Diagnosis of Continuous Systems

Matthew Daigle and **Xenofon Koutsoukos** and **Gautam Biswas**
Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
matthew.j.daigle,xenofon.koutsoukos,gautam.biswas@vanderbilt.edu

Abstract

Fault detection and isolation is a key component of any safety-critical system. Although diagnosis methods based on discrete event systems have been recognized as a promising framework, they cannot be easily applied to systems with complex continuous dynamics. This paper presents a novel approach for discrete event system diagnosis of continuous systems based on a qualitative abstraction of the measurement deviations from the nominal behavior. We systematically derive a diagnosis model, provide diagnosability analysis, and design a diagnoser. Our results show that the proposed approach is easily applicable and can be used for online diagnosis of abrupt faults in continuous systems.

Introduction

Fault detection and isolation (FDI) is a key component of safety-critical systems. Faults and degradations need to be quickly identified so corrective actions can be taken and catastrophic situations can be avoided. FDI methodologies can be categorized along several dimensions, such as model-based vs. signal-driven, online vs. offline, and continuous vs. discrete. Discrete event system (DES) methods are an important framework for diagnosis because of the significance of event-driven models in safety-critical systems, a well-developed theory that allows for systematic construction of diagnostic systems, and the computational efficiency it provides to enable online diagnosis for large systems.

Existing DES diagnosis methods (Sampath *et al.* 1996; 1995; Zad, Kwong, & Wonham 2003; Jiang & Kumar 2004) are based on detailed, automata-based models capturing both the nominal and faulty behavior traces of the system. Discrete event models have formed the basis for developing many practical diagnosis applications (Sampath *et al.* 1996; Kurien, Koutsoukos, & Zhao 2002; Benveniste *et al.* 2003; Chandra, Huang, & Kumar 2003), however, it is not clear how to systematically apply them to develop diagnosers for systems with complex continuous dynamics. Abstracting continuous dynamics requires quantization of the state space, resulting in large, nondeterministic models (Lunze 2000; Koutsoukos *et al.* 2000). Further, even if it is reasonable to abstract the nominal continuous behavior, developing the event-based behavior trajectories for fault conditions is very challenging. Faults in continuous dynamic systems are represented by changes in the system parameters, and therefore, quantization techniques must consider a high-dimensional state space and often complex nonlinear dynamics.

This paper presents a novel approach for DES diagnosis of continuous systems based on a qualitative abstraction of the measurement deviations from the nominal behavior. We describe a systematic method for generating a discrete event model of the system representing the faulty behaviors. The approach is derived from the TRANSCEND (Mosterman & Biswas 1999) methodology, a model-based approach to qualitative fault diagnosis in continuous systems. Starting from the continuous system model and the TRANSCEND approach to diagnosis, we use the concept of fault signatures combined with measurement orderings to build a discrete event diagnoser for isolating single faults in the system.

Specifically, the contribution of the paper is threefold: (*i*) we systematically construct a labeled transition system capturing the *fault language*, which, for each fault, describes all possible sequences of measurement deviations, (*ii*) we analyze the diagnosability of the system and design an event-based diagnoser, and (*iii*) we describe an implementation that improves the computational efficiency of the diagnoser. Diagnosis of component faults in an electric circuit is used throughout the paper to illustrate the approach.

Our approach to continuous systems diagnosis exploits the qualitative form of the fault transient created by abrupt deviations in component parameter values as well as the temporal ordering of measurement deviations, thereby generating event sequences (Daigle, Koutsoukos, & Biswas 2005; 2006; 2007). Since DES methods diagnose system failures based on sequences of observed events, there is a direct link between our diagnosis approach and DES approaches. We clarify this connection by first describing related work. We then present our modeling and analysis approach, the design of the diagnoser, and a comparison of our approach to DES methods.

Related Work

We formulate our approach to diagnosis of continuous systems into an event-based framework. DES diagnosis methods are based on observing system events and making inferences about the system state. The basic idea is that the occurrence of a fault will generate a unique sequence of observable events that will establish the presence of the fault.

(Sampath *et al.* 1996; 1995) describes a modeling and diagnosis framework for systems in the DES framework. A diagnoser based on the system model functions as an extended observer that provides estimates of the system state under nonfaulty and faulty conditions. (Zad, Kwong, & Wonham 2003) use a state-based approach, so the diagnoser determines system condition, rather than which failure events have occurred. (Jiang & Kumar 2004) present diagnosis of DES based on linear-time temporal logic (LTL) specifications. In this method, an individual diagnoser is designed for each fault specification, rather than constructing a single diagnoser for the global system.

To apply DES diagnosis approaches to continuous systems, the system models must be abstracted in some way. One method is to create a timed DES model. Such models typically include an additional observable event representing the tick of a global clock (Chen & Provan 1997; Zad, Kwong, & Wonham 1999). Diagnosis of timed DES has been investigated in (Chen & Provan 1997) as an extension of (Sampath *et al.* 1996) and in (Zad, Kwong, & Wonham 1999) as an extension of (Zad, Kwong, & Wonham 2003). Alternatively, a timed automaton model of the system can be used for diagnosis (Tripakis 2002). The approach of (Lunze 2000) develops the abstracted timed DES model through quantization. The continuous state space is partitioned and events defined for crossings of those partitions.

Chronicles are another method of modeling timed event traces in systems. Chronicles are patterns of event traces that include temporal constraints. They represent the possible timed evolutions of the system behaviors. Chronicles represent direct symptom to fault knowledge, so are therefore very efficient for online diagnosis (Bibas *et al.* 1996; Cordier & Dousson 2000). As events occur in the system, they are matched against known chronicles to determine which faults are present.

A different event-based abstraction for continuous systems can take measurement deviations as events. (Puig *et al.* 2005a) describes different methods for including timing information for fault isolation. One method is to set time bounds for symptom appearance as in (Kościelny 1995; Kościelny & Zakroczymski 2000). Another method is to consider the order of symptom appearance in what is called a *dynamic fault signature matrix*. In (Puig *et al.* 2005b), a fault diagnosis algorithm is described which uses this type of information. Results illustrate that the diagnostic precision is improved over methods that do not use timing information, and diagnostic error is improved over other methods like (Kościelny 1995). (Bayoudh, Travеé-Massuyès, & Olive 2006) take an alternative approach where the events are taken to be changes in binary residual values. This information is used to reconfigure the system in a way that increases diagnostic precision, because in some modes a residual would be 0 due to a fault, an in others it would be 1.

Background

Our diagnosis approach is based on the TRANSCEND methodology (Mosterman & Biswas 1999), a model-based approach to continuous systems diagnosis. Faults are represented as persistent, abrupt parameter changes in the system, modeled as a bond graph (Karnopp, Margolis, & Rosenberg 2000). We assume only single faults are likely to occur. We use an observer based on the system model to track the system and produce estimates of nominal behavior. When faults occur, they produce transients causing measurements to deviate from nominal behavior. To achieve robustness to sensor noise and model uncertainty, these deviations are analyzed to determine if they are statistically significant (Biswas *et al.* 2003). Significant deviations are used as they occur to isolate faults in the system. The diagnosis model, the temporal causal graph (TCG), is derived from the system model. It captures the propagation of fault effects on measurements and, therefore, is used to compute predicted effects of faults on measurements. By comparing predicted and observed effects on measurements, we can obtain diagnoses.

Measurement deviations are represented as qualitative $\pm$ values (above, below nominal), and are predicted as *fault signatures* using the TCG (Mosterman & Biswas 1999). A fault signature represents the qualitative value of zeroth- through kth-order derivative changes on a measurement due to a fault occurrence. Because only magnitude and slope can be reliably measured, we condense the signatures to the magnitude change symbol and the first nonzero derivative change, e.g., $00{-}{+}{-}$ becomes $0{-}$, and ${+}{-}{+}{-}{+}$ becomes ${+}{-}$. We can do this because higher-order changes will eventually manifest as first-order changes, and only the first change on a measurement provides discriminatory information (Manders *et al.* 2000). Therefore, we represent a fault signature for measurement m as an element of the set $\Sigma_m \triangleq \{m^{+-}, m^{-+}, m^{0+}, m^{0-}\}$[1]. The superscript indicates the observed deviation. The first symbol represents the immediate direction of change (a discontinuity) at fault occurrence and the second symbol represents the slope of the change after fault occurrence.

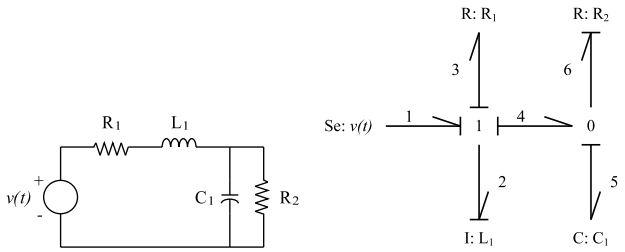
Definition 1 (Fault Signature)**.** *A* fault signature *for a fault f and measurement m is the qualitative effect of the occurrence of f on m, and is denoted by $\sigma_{f,m} \in \Sigma_{f,m}$, where $\Sigma_{f,m} \subseteq \Sigma_m$. We denote the set of all fault signatures for fault f as Σ_f.*

Relative measurement orderings define, with respect to a given fault, a partial order of measurement deviations, and are based on the intuition that some measurements deviate before others due to a fault. These are predicted using the TCG based on common temporal subpaths (Daigle, Koutsoukos, & Biswas 2005).

Definition 2 (Relative Measurement Ordering)**.** *Consider a fault f and measurements m_i and m_j. If f manifests in m_i before m_j then we define a* relative measurement ordering *between m_i and m_j for fault f, denoted as $m_i \prec_f m_j$. We denote the set of all measurement orderings for f as Ω_f.*

Throughout the paper we will illustrate the diagnosis methodology with a circuit example. Fig. 1(a) gives the

[1] In general, $\sigma_{f,m}$ may not be unique if the direction of change cannot be determined by qualitative propagation.



(a) Schematic. (b) Bond graph.

Figure 1: Circuit example.

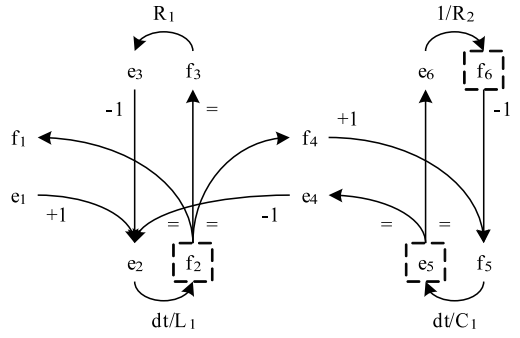


Figure 2: Temporal causal graph for the circuit. Measured variables are boxed.

schematic. We assume that our input voltage, $v(t)$, is constant and positive. The associated bond graph is given in Fig. 1(b). It models the elements of the circuit and the energy exchange between them (Karnopp, Margolis, & Rosenberg 2000). The derived TCG is given in Fig. 2. Relations between system variables are direct (+1) or inverse (-1) proportionality relations, component parameter values (e.g., R_1), or time-derivative effects (dt). The set of faults is assumed to be $F = \{R_1^+, R_1^-, R_2^+, R_2^-, C_1^+, C_1^-, L_1^+, L_1^-\}$, where the superscript indicates the direction of change of the parameter value. We define the measurement set as the current through L_1, the voltage across C_1, and the current through R_2, or $M = \{f_2, e_5, f_6\}$ in the bond graph model.

The fault signatures and relative measurement orderings for the circuit system are given in Table 1. For example, consider L_1^-. A decrease in L_1 will cause an immediate increase in f_2, because of the inverse relation implied in the TCG. Since all subsequent paths from f_2 to any other observed variable in the system contain some edge with a dt specifier (implying an integration), then deviations in these measurements will only be detected after f_2 deviates. Either e_5 or f_6 may deviate next. It cannot be determined which will deviate first because the path from e_5 to f_6 contains no integrals. The measurement deviations will not be abrupt because of the integration in the path from L_1 to the measurement, and the direction of change will be opposite that of f_2 because the -1 specifier in the path from f_2 to e_5 and f_6 indicates an inverse proportionality relationship.

Fault	f_2	e_5	f_6	Measurement Orderings
R_1^+	0−	0−	0−	$f_2 \prec e_5, f_2 \prec f_6$
R_1^-	0+	0+	0+	$f_2 \prec e_5, f_2 \prec f_6$
R_2^+	0−	0+	−+	$e_5 \prec f_2, f_6 \prec f_2, f_6 \prec e_5$
R_2^-	0+	0−	+−	$e_5 \prec f_2, f_6 \prec f_2, f_6 \prec e_5$
C_1^+	0+	−+	−+	$e_5 \prec f_2, f_6 \prec f_2$
C_1^-	0−	+−	+−	$e_5 \prec f_2, f_6 \prec f_2$
L_1^+	−+	0−	0−	$f_2 \prec e_5, f_2 \prec f_6$
L_1^-	+−	0+	0+	$f_2 \prec e_5, f_2 \prec f_6$

Table 1: Fault Signatures and Relative Measurement Orderings for the Circuit

Event-based Fault Modeling

We combine the notion of fault signatures and relative measurement orderings into an event-based framework. Essentially, for a specific fault, the combination of all fault signatures and relative measurement orderings yields all the possible ways a fault can manifest. We denote one of these possibilities as a *fault trace*.

Definition 3 (Fault Trace)**.** *A* fault trace *for a fault f, denoted by λ_f, is a string of length $\leq |M|$ that includes, for every $m \in M$ that will deviate due to f, a fault signature $\sigma_{f,m}$, such that the order of fault signatures satisfies Ω_f.*

Consider C_1^+. $\lambda_{C_1^+} = e_5^{-+} f_6^{-+} f_2^{0+}$ is a valid fault trace, but $\lambda_{C_1^+} = f_2^{0+} e_5^{-+} f_6^{-+}$ is not because the measurement deviation sequence does not satisfy $\Omega_{C_1^+}$. We group the set of all fault traces into a *fault language*, which can be represented concisely by a *labeled transition system* (LTS).

Definition 4 (Fault Language)**.** *The* fault language *of a fault f, denoted by L_f, is the set of all fault traces for f.*

Definition 5 (Labeled Transition System)**.** *A* labeled transition system *is a tuple $\mathcal{L} = (Q, q_o, \Sigma, \delta)$ such that: Q is a set of states, $q_o \in Q$ is an initial state, Σ is a set of labels, and $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation.*

To systematically construct the LTS representation of a fault language, called a *fault model*, we can represent each fault signature and each relative measurement ordering as an LTS, and then compose all the information. Each fault signature $\sigma_{f,m}$ can be represented as an LTS, shown to the left of Fig. 3. It consists of only the single event corresponding to the fault signature[2]. Also, each relative measurement ordering, $m_i \prec_f m_j$, with associated signatures σ_{f,m_i} and σ_{f,m_j}, can be represented as an LTS, shown to the right of Fig. 3. It consists of the two associated signatures in the determined ordering.

Lemma 1. *The LTS representation of a fault language L_f for fault f, denoted by $\mathcal{L}_f$, is the synchronous product of the individual LTS for all $\sigma_{f,m} \in \Sigma_f$ and all $m_i \prec_f m_j \in \Omega_f$, where the alphabets for each LTS are taken to be the events contained in the LTS.*

[2]If $\sigma_{m,f}$ is not unique, multiple edges for each possibility are needed going from the first state of the LTS to the final state.

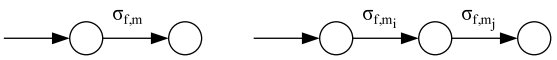


Figure 3: Fault signature LTS representation (left) and relative measurement ordering LTS representation (right).

Proof. Since the synchronous product must obey all individual ordering constraints and includes all measurement deviation events for the fault, it produces all valid measurement deviation sequences and no others. □

Lemma 2 (Distinguishability)**.** *A fault f_i is* distinguishable *from a fault f_j, denoted by $f_i \nsim f_j$, if $(\forall\, \lambda_{f_i} \in L_{f_i}, \lambda_{f_j} \in L_{f_j})\ (\neg\exists\, \lambda)\ \lambda_{f_i}\lambda = \lambda_{f_j}$.*

Proof. Two faults are distinguishable if it is not possible for them to manifest in the measurements in the same way. Since a fault language represents all possible measurement deviation sequences for a particular fault, if one fault exhibits a trace that is a substring of another fault, then the faults cannot be distinguished. Otherwise, they cannot manifest in the same way and are distinguishable. □

Depending on how they actually manifest in the system however, two faults which are indistinguishable may be discriminated if fault f_i occurs and manifests in a way that it is not possible for fault f_j to manifest, i.e, $\lambda_{f_i} \notin L_{f_j}$. Distinguishability is, therefore, a conservative notion. To design diagnosers, we look for the notion of *diagnosability*, based on the notion of distinguishability.

Lemma 3 (Diagnosability)**.** *A system is* single fault diagnosable *if $(\forall f_i, f_j \in F) f_i \neq f_j \implies f_i \nsim f_j$.*

Proof. A system is diagnosable if each possible fault trace is consistent with a unique fault. If two faults are distinguishable, then they cannot manifest in the same way. Therefore, if all pairs of faults are distinguishable, then a given fault trace cannot be consistent with the more than one fault. Therefore, the fault trace corresponds to a unique fault, so the system is diagnosable. □

Diagnoser Design

The notion of diagnosability is used in building correct diagnosers. To guarantee unique diagnosis of every fault, a system must be diagnosable. We now describe a method to systematically create such a diagnoser, but first, we define formally a *diagnosis* and a *diagnoser*.

Definition 6 (Diagnosis)**.** *A* diagnosis *$d \subseteq F$ is a set of faults consistent with the observations.*

Definition 7 (Diagnoser)**.** *A* diagnoser *is a tuple $\mathcal{D} = (Q, q_o, \Sigma, \delta, D, Y)$ such that: Q is a set of states, $q_o \in Q$ is an initial state, Σ is a set of labels, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $D \subseteq C$ is a set of diagnoses, and $Y : Q \to D$ is a diagnosis map.*

A diagnoser is a LTS extended by a set of diagnoses and a diagnosis map. Similar to the LTS of a fault, the labels correspond to measurement deviations. Associated with the states are diagnoses, i.e., the set of possible faults for the measurement deviations seen thus far.

Algorithm 1 $\mathcal{D} \leftarrow$ `CreateDiagnoser`$(\mathcal{D}_1, \mathcal{D}_2)$

```
Q ← ∅, δ ← ∅, D ← ∅, Σ ← Σ1 ∪ Σ2
q_o ← (q_o1, q_o2), Y(q_o) ← ∅, Q_pend ← {q_o}
while Q_pend ≠ ∅ do
  (q1, q2) ← pop(Q_pend)
  for all σ_m ∈ Σ do
    if m ∉ H((q1, q2)) then
      if δ1(q1, σ_m) and δ2(q2, σ_m) then
        q' ← (δ1(q1, σ_m), δ2(q2, σ_m))
        h ← Y(δ1(q1, σ_m)) ∪ Y(δ2(q2, σ_m))
      else if δ1(q1, σ_m) then
        q' ← (δ1(q1, σ_m), q2)
        h ← Y(δ1(q1, σ_m))
      else if δ2(q2, σ_m) then
        q' ← (q1, δ2(q2, σ_m))
        h ← Y(δ2(q2, σ_m))
      else
        q' ← ∅
        h ← ∅
      if q' ≠ ∅ then
        if Y((q1, q2)) = ∅ then
          d ← h
        else
          d ← Y((q1, q2)) ∩ h
        if d ≠ ∅ then
          Q ← Q ∪ {q'}
          H(q') ← H((q1, q2)) ∪ {m}
          δ((q1, q2), σ_m) ← q'
          D ← D ∪ {d}
          Y(q') ← d
          if q' ∉ Q_pend then
            push(Q_pend, q')
```

The diagnoser construction procedure is shown as Algorithm 1. It is described as combining two diagnosers, but can be easily be modified to combine k diagnosers simultaneously. Diagnosers are constructed by incrementally composing subdiagnosers, i.e., a diagnoser for a set of faults F_i is composed with a diagnoser for a set of faults F_j to create a new diagnoser for $F_i \cup F_j$. Initially, we begin with diagnosers for singleton fault sets. These are constructed using the individual fault models. For a single fault f, we augment $\mathcal{L}_f$ to form $\mathcal{D}_f$ by constructing the diagnosis map as mapping every state except the initial state to $\{f\}$. The initial state is mapped to the empty diagnosis, $\varnothing$, because until a measurement deviation is observed, we assume the system is operating nominally. The diagnosers corresponding to the individual faults of the circuit are shown in Fig. 4.

The construction algorithm operates by tracing paths in the two given diagnosers. If the same event label is available in both current states, then we advance in both machines, i.e., $(q_1, q_2) \xrightarrow{\sigma} (\delta(q_1, \sigma), \delta(q_2, \sigma))$. Otherwise, we advance in only one, e.g., if σ can only be taken from q_1, then , $(q_1, q_2) \xrightarrow{\sigma} (\delta(q_1, \sigma), q_2)$. However, if the measurement associated with σ has already deviated along the cur-

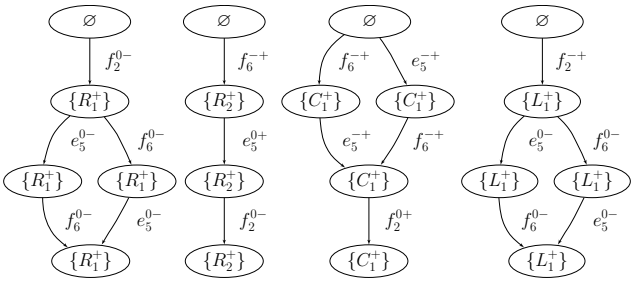


Figure 4: Diagnosers for the individual faults of the circuit. The diagnosers for decreases in the parameter values the same except for a reversal in the signs.

rent path (tracked using H), $\delta((q_1, q_2), \sigma)$ is set to $\varnothing$, because measurement deviations are only detected once per measurement. This also occurs if the computed diagnosis for the new state, d, is empty, because this means the current sequence of measurement deviations is inconsistent with the single fault assumption.

The diagnosis for the new state is formed by composing the current diagnosis with the hypothesis set. The hypothesis set, h, is the set of faults consistent with the current event. It is formed as the union of the diagnoses of the diagnoser states advanced to via σ. The new diagnosis for the composed diagnoser state is constructed as the intersection of the current diagnosis and the hypothesis set. For example, if $\{f_i, f_j\}$ is the current diagnosis and the hypothesis set is $\{f_i\}$ then the new diagnosis is $\{f_i\}$, which means that only f_i is consistent with the current event sequence.

The final composed diagnoser for the circuit is illustrated in Fig. 5. For example, consider the fault trace $f_6^{-+} e_5^{0+} f_2^{0-}$. For f_6^{-+} occurring as the first measurement deviation, only C_1^+ or R_2^+ could have occurred, given the known fault signatures and relative measurement orderings. Therefore, the new diagnosis is $\{C_1^+, R_2^+\}$. For e_5^{0+} occurring next, of our current faults, only R_2^+ is consistent, therefore our new diagnosis is the intersection of $\{C_1^+, R_2^+\}$ and $\{R_2^+\}$, which is $\{R_2^+\}$. At this point we obtain a unique fault. The only possible measurement deviation from here is f_2^{0-} which must be consistent still with $\{R_2^+\}$.

Theorem 1. *The diagnoser constructed by Algorithm 1 for fault sets F_1 and F_2 represents all valid single fault traces for the faults in F_1 and F_2 and associates correct diagnoses with the states.*

Proof. By definition, the diagnoser for a single fault f is correct because it represents L_f, so represents all possible fault traces of f, and every state (except the initial state) of $\mathcal{L}_f$ is consistent with f occurring. Assume that diagnosers $\mathcal{D}_1$ and $\mathcal{D}_2$ are correct. Then they represent all possible fault traces for fault sets F_1 and F_2, respectively. At the initial state, if an event σ happens which can only happen in one of the diagnosers, $\mathcal{D}_i$, then the diagnosis is $Y_i(\delta(q_{oi}, \sigma))$, because it must be consistent with faults in F_i that are consistent with σ. If σ can occur in both diagnosers, then the diagnosis is $Y_1(\delta_1(q_{o1}, \sigma)) \cup Y_2(\delta_2(q_{o2}, \sigma))$ because either a fault in F_1 occurred or a fault in F_2 occurred, and the diagnosis must be consistent with any fault in $F_1 \cup F_2$ consistent with σ. Assume that for a given $(q_1, q_2) \neq q_o$ the diagnoses are correct for the event sequences leading up to (q_1, q_2). Then if an event σ happens which can only happen in one of the diagnosers, $\mathcal{D}_i$, then the diagnosis is $Y((q_1, q_2)) \cap Y_i(\delta_i(q_i, \sigma))$, because it must consistent with the previous diagnosis faults in F_i consistent with σ. If σ can occur in both diagnosers, then the diagnosis is $Y((q_1, q_2)) \cap (Y_1(\delta_1(q_1, \sigma)) \cup Y_2(\delta_2(q_2, \sigma)))$ because it must be consistent with the previous diagnosis and faults in $F_1 \cup F_2$ consistent with σ. Therefore, for any state q, $\delta(q, \sigma)$ has a correct diagnosis. So, for any two diagnosers, the resulting diagnoser is correct. □

The diagnoser for the circuit example, shown in Fig. 5, illustrates certain properties of our approach. Since all the leaves have diagnoses with a unique fault, then the system is diagnosable. Any possible sequence of measurement deviations corresponding to a single fault occurring are captured in the diagnoser, and lead to unique diagnoses, therefore the system is diagnosable. We can also see that a unique diagnosis is obtained after only two of the three measurements deviate, therefore one measurement is redundant for single fault diagnosis of the selected faults.

Online Diagnoser Implementation

This event-based diagnosis framework leads to three different implementations of the onlnie diagnosis approach that trade off space and time complexity.

Implementation as a global LTS Time complexity is in favor of the precomputed global diagnoser (Fig. 5). It needs only to wait for measurement deviations to occur, transition to the next state, and output the current diagnosis associated with the state. Using appropriate data structures, these operations can be achieved in constant time.

The complete diagnoser has, in the worst case, $O(|M|!)$ possible fault traces, and thus $O(|M|!)$ states. Therefore this approach will not be space-efficient, in general. If many temporal orderings exist, then the number of possible fault traces reduces significantly, and the global diagnoser approach may be feasible.

Implementation as an LTS for each fault In this approach, we only precompute the individual fault diagnosers (Fig. 4). Each fault has $O(|M|!)$ possible fault traces, but if there are many temporal orderings, this may also be reduced for many faults.

For online diagnosis, each diagnoser is traced simultaneously. When a diagnoser becomes blocked, i.e., there is no available event to to take from the current state, then it is no longer tracked, because it is no longer consistent with the observed measurement deviations. The candidate set is formed by taking the union of the faults in the current states of each active diagnoser, i.e., those faults that are still consistent with the observed measurement deviations. This operation is simply $O(|F|)$ in time.

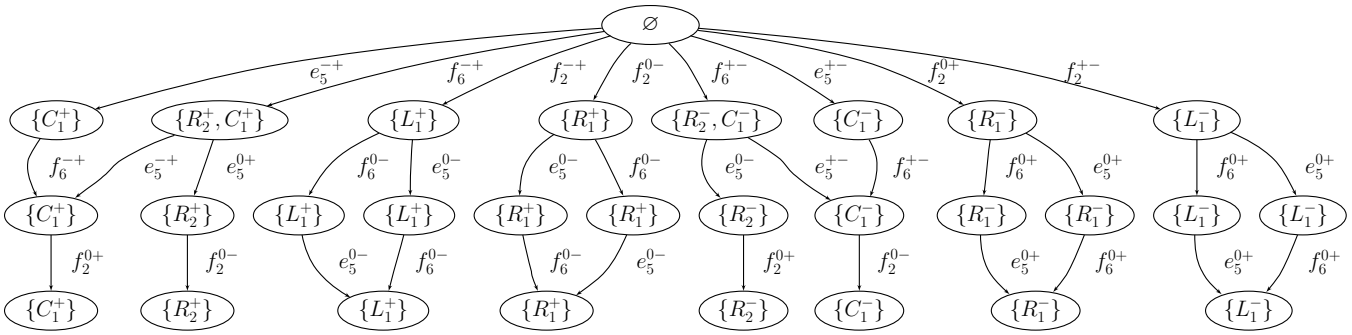


Figure 5: Single fault diagnoser for the circuit.

Implementation without explicit event fault models If the faults are very temporally constrained, then any of the two above approaches should be both space-efficient and time-efficient. If few orderings are available, then the diagnosers approach size $O(|M|!)$, therefore these approaches may not be feasible given the space requirements of the system. For diagnosis without using LTS-based diagnosers, we store only the fault signatures and relative measurement orderings for each fault (Table 1), requiring $O(|F||M|^2)$ space.

Given a current diagnosis of d_{i-1} and an event σ_i occurring, we can check which faults are consistent with σ_i. The hypothesis set h_i consists of those faults. If $i = 1$, then the new diagnosis d_i is simply h_i. Otherwise, the new diagnosis must be consistent with d_{i-1} and with the new information, i.e., $d_i = d_{i-1} \cap h_i$. Therefore, given d_{i-1}, the new diagnosis can be computed simply as the subset of faults in d_{i-1} consistent with σ_i. This corresponds to only constructing the *path* of the global diagnoser relating to the particular fault trace we are observing.

For online diagnosis, we form the hypothesis set corresponding to the current measurement deviation by looking through the fault signatures and measurement orderings, thus taking $O(|F||M|^2)$ time. We then compute the new diagnosis, which is a function of the size of the current diagnosis and the current hypothesis set. In the worst case the hypothesis set consists of all faults, so it is $|F|$ in size. A diagnosis can be as large as $|F|$ also. The intersection of the diagnosis and hypothesis set then takes at worst $O(|F|)$ time. In practice, this time complexity is reduced because as measurements deviate, less faults are being considered.

Discussion and Comparison

Though diagnosis using ordered event sequences is performed similarly in all approaches, the main contrast between existing methods and our approach is the abstraction used to generate the DES models. Most of the traditional DES approaches (Sampath *et al.* 1996; 1995; Zad, Kwong, & Wonham 2003; Jiang & Kumar 2004; Rozé & Cordier 2002; Baroni *et al.* 1999; Benveniste *et al.* 2003; Chen & Provan 1997; Zad, Kwong, & Wonham 1999) assume models created by human experts, and others assume subsystem models created by experts that are then composed to form the global model. The DES model and all its faulty behavior is assumed to be given. To derive a discrete event model for a continuous system, the continuous dynamics must be abstracted in some way. In quantization approaches, e.g., (Lunze 2000), the state space is quantized. This results in several problems. First, the model is, except in trivial cases, inherently nondeterministic, which degrades the performance and increases the computational requirements of diagnosis algorithms. Second, the resulting model is very large. The finer-grained the quantization and the greater the range of possible inputs, the larger and more complex the model will be. To use the quantization approach, faults have to be quantized as well, according to their magnitude and other characteristics. In addition, if faults are possible at any state of the system (as is usually the case), then the DES model becomes larger still, and the number of states explodes.

In our approach, however, faults are represented as parameter changes in the nominal model of the system. As a result, the system model represents both nominal and faulty behavior in a very concise way. From this model (the bond graph model), we can systematically derive the diagnosis model (Mosterman & Biswas 1999), i.e., the TCG, generate fault signatures and measurement orderings, and extract from this information a DES model of the system with respect to faulty behavior. This greatly reduces the burden of the modeling task, as well as providing a systematic framework for deriving the faulty behavior. Approaches such as (Puig *et al.* 2005a) or chronicles do not provide a way to systematically obtain this information from a system model except for very specific applications (Dousson 1996). Additionally, our approach is not dependent on fault magnitude because we are only concerned with the qualitative form of the measurement deviations.

Our approach can be viewed as a *qualitative* abstraction of the observed behavior from the nominal behavior. We model only the faulty behavior relevant to diagnosis, so there are three qualitative states for each measurement: *above nominal*, *at nominal*, and *below nominal*. Measurement deviations directly indicate the presence of a fault. The only state of the diagnoser modeling nominal behavior is the initial state, in which no measurement deviations have been observed. Our nominal behavior is defined through an observer (Manders *et al.* 2000), which is the best way to track a con-

tinuous system with noise. Therefore, faults can be detected very quickly, unlike in quantization approaches, where the fault detection time will depend on the level of quantization, whereas in our approach, fault detection time is a function of noise only. The tasks of tracking nominal behavior and fault isolation are separated so that the diagnoser is concerned only with faulty behavior.

In a way, our abstraction can be viewed as qualitative deviation models (Struss 2004). However, the TCG represents this information in a concise manner, and reasoning with the TCG is very efficient. Further, the TCG is based directly on the system behavior defined by the continuous state equations, so captures complex dynamics and interactions. Generating the predictions in the form of qualitative deviations from the TCG is automatic. We also include discontinuity detection for increased discriminatory ability which is not taken into account by traditional qualitative deviation models.

Other continuous diagnosis approaches that use temporal information (Kościelny 1995; Kościelny & Zakroczymski 2000; Puig *et al.* 2005a; 2005b; Bayoudh, Travеé-Massuyès, & Olive 2006) are based on analytical redundancy methods. These methods are difficult to apply to nonlinear systems and multiplicative faults. In addition, they do not consider the sign of the residual or whether a discontinuity was present in the signal to help isolate faults. The use of time bounds in some of these approaches and in chronicle approaches (Bibas *et al.* 1996; Cordier & Dousson 2000) is also difficult to use for continuous systems. It requires significant analysis to obtain the time bounds, which are often conservative, and assume bounds on fault magnitude, which cannot be made in general.

Because we are working in an event-based framework, the notions of fault traces, fault languages, distinguishability, and diagnosability bear a resemblance to those defined in the DES framework. The notion of a fault trace is similar to the notion of a fault signature defined for DES in (Cordier, Travé-Massuyès, & Pucel 2006), where it is defined as a string (of finite or infinite length) that contains a fault event. Diagnosability can then be defined in terms of ensuring no two faults have the same signature (in our case, the same trace). This is the situation in any modeling framework, because in general, faults can only be distinguished if they manifest in different ways, in whatever way that is represented by the model. Our approach separates out the fault effects and analyzes them separately. The individual fault diagnosers generated by our approach are also similar to chronicles and the individual diagnoser of (Jiang & Kumar 2004). The global diagnoser bears resemblance to (Sampath *et al.* 1996; Zad, Kwong, & Wonham 2003).

Another advantage of operating within an event-based framework is that the model created by our qualitative abstraction approach can be used with any of the other DES diagnosis approaches. However, since in our approach, we essentially abstract out events corresponding to nominal behavior, we obtain a direct mapping of finite event sequences to faults. Consequently, a simpler diagnosis approach than those defined for general DES models can be employed. Because of this, the diagnoser is simpler, and also, does not need to be computed at design time, which improves considerably space-efficiency, because the particular path corresponding to the given measurement deviations can be constructed online efficiently.

Conclusions

We have presented an event-based approach to diagnosis of single abrupt faults in continuous systems. We use a qualitative abstraction from nominal behavior. The approach results in systematic generation of event-based fault models and diagnosers, based on qualitative fault signatures and temporal orderings of measurement deviations. Current and future work is addressing multiple fault diagnosis and extending our hybrid systems diagnosis algorithms (Narasimhan & Biswas 2007) under this framework.

Acknowledgments

This work was supported in part by NSF grant CNS-0615214, NASA USRA grant 08020-013, and NASA NRA grant NNX07AD12A.

References

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

Bayoudh, M.; Travеé-Massuyès, L.; and Olive, X. 2006. Hybrid systems diagnosability by abstracting faulty continuous dynamics. In *Proceedings of the 17th International Principles of Diagnosis Workshop*, 9–15.

Benveniste, A.; Fabre, E.; Haar, S.; and Jard, C. 2003. Diagnosis of asynchronous discrete-event systems: A net unfolding approach. *IEEE Transactions on Automatic Control* 48(5):714–727.

Bibas, S.; Cordier, M.-O.; Dague, P.; Dousson, C.; Lvy, F.; and Roz, L. 1996. Alarm driven supervision for telecommunication network: I – off-line scenarios generation. *Annales des Telecommunications* 51(910):493–500.

Biswas, G.; Simon, G.; Mahadevan, N.; Narasimhan, S.; Ramirez, J.; and Karsai, G. 2003. A robust method for hybrid diagnosis of complex systems. In *Proceedings of the 5th Symposium on Fault Detection, Supervision and Safety for Technical Processes*, 1125–1131.

Chandra, V.; Huang, Z.; and Kumar, R. 2003. Automated control synthesis for an assembly line using discrete event system control theory. *IEEE Trans. on Systems, Man and Cybernetics, Part C* 33(2):284–289.

Chen, Y.-L., and Provan, G. 1997. Modeling and diagnosis of timed discrete event systems – a factory automation example. In *Proc. of the American Control Conf.*, 31–36.

Cordier, M.-O., and Dousson, C. 2000. Alarm driven monitoring based on chronicles. In *Proceedings of the 4th Symposium on Fault Detection Supervision and Safety for Technical Processes (Safeprocess 2000)*, 286–291.

Cordier, M.-O.; Travé-Massuyès, L.; and Pucel, X. 2006. Comparing diagnosability in continuous and discrete-event

systems. In *Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06)*, 55–60.

Daigle, M.; Koutsoukos, X.; and Biswas, G. 2005. Relative measurement orderings in diagnosis of distributed physical systems. In *43rd Annual Allerton Conference on Communication, Control, and Computing*, 1707–1716.

Daigle, M.; Koutsoukos, X.; and Biswas, G. 2006. Distributed diagnosis of coupled mobile robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, 3787–3794.

Daigle, M. J.; Koutsoukos, X. D.; and Biswas, G. 2007. Distributed diagnosis in formations of mobile robots. *IEEE Transactions on Robotics* 23(2):353–369.

Dousson, C. 1996. Alarm driven supervision for telecommunication network: Ii – on-line chronicle recognition. *Annales des Telecommunications* 51(910):501–508.

Jiang, S., and Kumar, R. 2004. Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Transactions on Automatic Control* 49(6):934–945.

Karnopp, D. C.; Margolis, D. L.; and Rosenberg, R. C. 2000. *Systems Dynamics: Modeling and Simulation of Mechatronic Systems*. New York: John Wiley & Sons, Inc.

Kościelny, J. M., and Zakroczymski, K. 2000. Fault isolation method based on time sequences of symptom appearance. In *Proceedings of IFAC SafeProcess 2000*.

Kościelny, J. M. 1995. Fault isolation in industrial processes by the dynamic table of states method. *Automatica* 31(5):747–753.

Koutsoukos, X.; Antsaklis, P.; Stiver, J.; and Lemmon, M. 2000. Supervisory control of hybrid systems. *Proceedings of IEEE* 88(7):1026–1049.

Kurien, J.; Koutsoukos, X.; and Zhao, F. 2002. Distributed diagnosis of networked embedded systems. In *Proceedings of the 13th International Workshop on Principles of Diagnosis (DX-02)*, 179–188.

Lunze, J. 2000. Diagnosis of quantized systems based on a timed discrete-event model. *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 30(3):322–335.

Manders, E.-J.; Narasimhan, S.; Biswas, G.; and Mosterman, P. 2000. A combined qualitative/quantitative approach for fault isolation in continuous dynamic systems. In *SafeProcess 2000*, volume 1, 1074–1079.

Mosterman, P., and Biswas, G. 1999. Diagnosis of continuous valued systems in transient operating regions. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 29(6):554–565.

Narasimhan, S., and Biswas, G. 2007. Model-based diagnosis of hybrid systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 37(3):348–361.

Puig, V.; Quevedo, J.; Escobet, T.; and Pulido, B. 2005a. On the integration of fault detection and isolation in model-based fault diagnosis. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05)*, 227–232.

Puig, V.; Schmid, F.; Quevedo, J.; and Pulido, B. 2005b. A new fault diagnosis algorithm that improves the integration of fault detection and isolation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, 3809–3814.

Rozé, L., and Cordier, M.-O. 2002. Diagnosing discrete-event systems: Extending the "diagnoser approach" to deal with telecommunication networks. *Discrete Event Dynamic Systems: Theory and Applications* 12:43–81.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology* 4(2):105–124.

Struss, P. 2004. Deviation models revisited. In *Working Papers of the 18th International Workshop on Qualitative Reasoning*.

Tripakis, S. 2002. Fault diagnosis for timed automata. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT02)*, volume 2469 of *Lecture Notes in Computer Science*, 205–221. Springer.

Zad, S. H.; Kwong, R. H.; and Wonham, W. M. 1999. Fault diagnosis in timed discrete-event systems. In *Proc. of the 38th Conference on Decision and Control*, 1756–1761.

Zad, S. H.; Kwong, R.; and Wonham, W. 2003. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Transactions on Automatic Control* 48(7):1199–1212.

Dynamic domain abstraction through meta-diagnosis

Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
dekleer@parc.com

Abstract

One of the most powerful tools designers have at their disposal is abstraction. By abstracting from the detailed properties of a system, the complexity of the overall design task becomes manageable. Unfortunately, faults in a system need not obey the neat abstraction levels of the designer. This paper presents an approach for identifying the abstraction level which is as simple as possible yet sufficient to address the task at hand. The approach chooses the desired abstraction level through applying model-based diagnosis at the meta-level, i.e., to the abstraction assumptions themselves.

1 Introduction

Of the many tools designers have at their disposal, abstraction is one of the most powerful. By abstracting from the detailed properties of a system, the complexity of the overall design task becomes manageable. For example, a computer engineer can focus on the logic level without concern for the properties of the individual transistors which make up a particular gate, and a chip designer can layout a chip without being concerned with the fabrication steps needed to construct it. Abstraction allow designers to partition concerns into independent black boxes and is one of the most important ideas underlying the design of modern technology.

Unfortunately, faults in a system need not obey the neat abstraction levels of the designer. A fault in a few transistors can cause an Intel Pentium processor to generate an ocasional incorrect floating point result. To understand this fault requires transcending the many abstraction levels between software and hardware. A PC designer can focus on functional layout without being concerned about the physical layout and its thermal properties. However, a technician must determine that the processor crashed because dust sucked into the processor fan clogged the heatsink and allowed the processor temperature to rise to such a dangerous level that the PC automatically shut down. As a consequence diagnostic reasoning is inherently messy and complex, as it involves crossing abstraction boundaries never contemplated by the designers.

Existing model-based reasoning has addressed a number of types of abstraction.

- Range abstraction. The ranges of variables are abstracted, e.g., instead of a continuous quanity it might be represented by the qualitative values of $-$, 0 or $+$. [Struss, 1991b; 1991a; Sachenbacher and Struss, 2005; Torta and Torasso, 2003]
- Structural abstraction. Groups of components are abstracted to form hierarchies [Chittaro and Ranon, 2004; Hamscher, 1990].
- Model selection. Approaches to choosing among a collection of hand-constructed models [Addanki *et al.*, 1989; Falkenhainer and Forbus, 1991]

In this paper we present a new type of abstraction (*domain* abstraction): changing the physical principles which underlie models, such as moving from the $0/1$ level to currents and voltages and providing a systematic approach to choosing the appropriate domain for the diagnostic task.

Choosing the right domain abstraction level requires balancing two opposing desiderata. Reasoning at the highest abstraction level is the simplest. Unfortunately, it may be inadequate to analyze or troubleshoot the system. Instead, the system needs to be analyzed at a more detailed level. On the other hand, reasoning at too low of a level can require enormously more computational resources and difficult to obtain parameters, and it generates more complicated analyses. As Albert Einstein reportedly said: "Make everything as simple as possible, but not simpler."

Technicians expect that systems are non-intermittent and that the schematic is an accurate description of the physical system. Consider the simple analog circuit of Figure 1. Suppose a technician measures the current to be 0 ampere (1 is expected), which leads to an inference that the resistor is faulty, but repeating the measurement gives 1 ampere. Either the resistor is intermittent or there is a fault in the connections. The technician must change abstraction level to diagnose this system further by, for example, checking the connections or further tests on resistor R itself.

Consider a circuit of three logic inverters in sequence, with its output fedback to its input (Figure 2). At the usual gate level of analysis, an inverter simply complements its input. This circuit has no inputs, so we need to consider the possible values at the connecting nodes. Assume the input to inverter A is 0. Its output must be 1. The output of B must be 0. The output of D must be 1. This is impossible, as we assumed

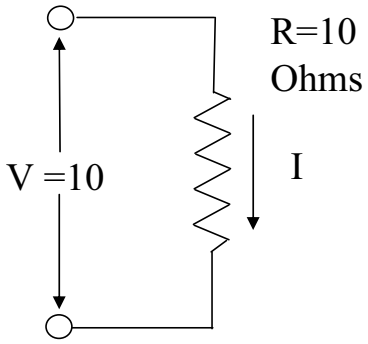


Figure 1: Simple analog diagnosis problem.

it was 0. Conversely, assume the input to inverter A is 1. Its output must be 0. The output of B must be 1. The output of D must be 0. This is impossible, as we assumed it was 1. Therefore, the input of inverter A can neither be 0 or 1. Also, the inputs of inverters B or D cannot be 0 or 1. Somehow the circuit is contradictory when modeled as logic gates. Thus, one of the components A, B or D must be faulted. Suppose the technician chooses to systematically remove and check each of these three components for proper functioning. If each component is confirmed to be correct, the technician has encountered an impasse.

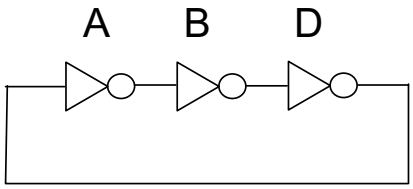


Figure 2: A very simple circuit which yields a contradiction when analyzed as combinational logic; yet its a perfectly reasonable fault-free circuit with well-defined behavior.

Analyses that result in contradictions are the most important indicator that the level of abstraction used is too simplistic. In this paper we present a general reasoner which automatically descends abstraction layers to perform needed analyses, and which does not descend abstraction levels needlessly. This approach is broadly applicable, but we explore these ideas in the context of digital circuits with messier models that include failures in connections, intermittents, and temporal behaviors.

2 Meta-Diagnosis

Figure 3 characterizes the basic architecture of a typical model-based, component-based diagnosis engine. Given the component topology (e.g., the schematic for analog circuits), component models (e.g., resistors obey ohm's law) and observations (e.g., the voltage across resistor $R6$ is 4 volts), it computes diagnoses which explain all the observations. Observations inconsistent with expectations guide the discovery of diagnoses. When the MBD engine can find no diagnoses it signals a failure.

Suppose we need to troubleshoot the circuit of Figure 2. Most diagnosis systems would immediately conclude that some subset of the components $\{A, B, D\}$ is faulted. However, testing each inverter individually demonstrates that all the components are good. As a consequence, most algorithms would report an unresolvable contradiction.

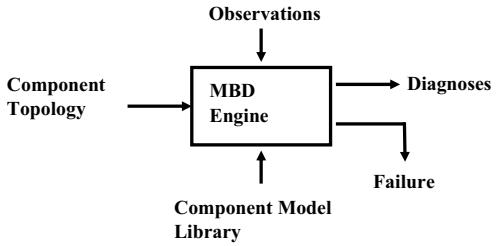


Figure 3: Basic architecture of a model-based diagnosis engine.

The architecture of Figure 4 includes two model-based diagnosis engines. The top model is used to identify the best abstraction level, and the bottom model performs the actual system diagnosis. This composite architecture has the same inputs as the basic architecture with one additional input: the abstraction library. The 'Applicable Models' module identifies all the applicable abstractions for the component topology. The 'Modeler' module uses the preferred meta-diagnosis to construct conventional model-based diagnosis models using the 'Component Model Library.'

Consider the example of Figure 2. The component topology is simply the circuit schematic as before. The system observations are as before (e.g., the output of A is 1). The component model library is will contain different models for gate behavior (e.g., boolean, analog, thermal, temporal, etc.). The new input, the abstraction library, is the set of all possible abstractions. Instead of a usual component topology, the abstraction MBD engine is provided width a set of possible abstractions applicable to the given system to be diagnosed. Initially, there are no meta-observations, so the preferred diagnosis is the one at the most abstract level (analogous to all components working). Therefore, the domain MBD engine will perform diagnosis in the usual way with the most abstract models. Suppose each gate is physically checked, leading to the observations that A, B and D are working correctly. The domain model-based engine now fails as it has found an unresolvable contradiction. This invokes the abstraction MBD engine as an observation. As analysis proceeds, the preferred meta-diagnosis will descend abstraction levels. For the purposes of this paper, the preferred meta-diagnosis is one minimal cardinality meta-diagnosis.

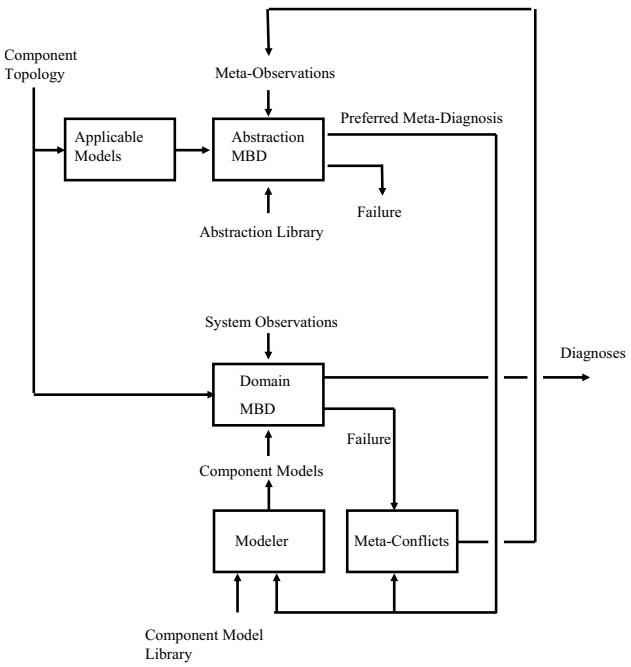


Figure 4: Architecture of an abstraction-based, model-based diagnosis engine.

3 Formalization

This section summarizes the formal framework for model-based diagnosis we use in the rest of the paper [de Kleer and Williams, 1987; de Kleer *et al.*, 1992]. In order to distinguish between domain and abstraction AB literals, we state the usual framework in terms of domain AB_d literals.

Definition 1 *A system is a triple (SD,COMPS, OBS) where:*

1. *SD, the system description, is a set of first-order sentences.*
2. *COMPS, the system components, is a finite set of constants.*
3. *OBS, a set of observations, is a set of first-order sentences.*

In Figure 3 *SD* is the component topology and component model library, and *COMPS* is the set of components in the component topology.

Definition 2 *Given two sets of components Cp and Cn define $\mathcal{D}_d(Cp, Cn)$ to be the conjunction:*

$$\Big[\bigwedge_{c\in Cp} AB_d(c)\Big] \wedge \Big[\bigwedge_{c\in Cn} \neg AB_d(c)\Big].$$

Where $AB_d(x)$ represents that the component x is ABnormal (faulted).

A diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 3 *Let $\Delta \subseteq COMPS$. A diagnosis for (SD,COMPS,OBS) is $\mathcal{D}_d(\Delta, COMPS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \{\mathcal{D}_d(\Delta, COMPS - \Delta)\}$$

Definition 4 *An AB_d-literal is $AB_d(c)$ or $\neg AB_d(c)$ for some $c \in COMPS$.*

Definition 5 *An AB_d-clause is a disjunction of AB_d-literals containing no complementary pair of AB_d-literals.*

Definition 6 *A conflict of (SD,COMPS,OBS) is an AB_d-clause entailed by $SD \cup OBS$.*

3.1 Formalizing abstraction

The abstraction MBD is defined analogously:

Definition 7 *An abstraction system is a triple (SD,ABS, OBS) where:*

1. *SD, constraints among possible abstractions, is a set of first-order sentences.*
2. *ABS, the applicable abstractions, is a finite set of constants.*
3. *OBS, a set of meta-observations, is a set of first-order sentences.*

Definition 8 *Given two sets of abstractions Cp and Cn define $\mathcal{D}_a(Cp, Cn)$ to be the conjunction:*

$$\Big[\bigwedge_{c\in Cp} AB_a(c)\Big] \wedge \Big[\bigwedge_{c\in Cn} \neg AB_a(c)\Big].$$

Where $AB_a(x)$ represents that the abstraction x is ABnormal (cannot be used).

A meta-diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 9 *Let $\Delta \subseteq ABS$. A meta-diagnosis for (SD,ABS,OBS) is $\mathcal{D}_a(\Delta, ABS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \{\mathcal{D}_a(\Delta, ABS - \Delta)\}$$

Definition 10 *An AB_a-literal is $AB_a(c)$ or $\neg AB_a(c)$ for some $c \in ABS$.*

Definition 11 *An AB_a-clause is a disjunction of AB_a-literals containing no complementary pair of AB_a-literals.*

Definition 12 *A meta-conflict of (SD,ABS,OBS) is an AB_a-clause entailed by $SD \cup OBS$.*

4 Example of a lattice of models

To illustrate these ideas we use 3 axes of abstraction:

- Model of connections as in [de Kleer, 2007b] which is an improvement over [Böttcher, 1995; Böttcher *et al.*, 1996].
- Model of non-intermittency [Raiman *et al.*, 1991] or intermittency [de Kleer, 2007a]
- Model of time [de Kleer, 2007c].

The corresponding AB_a literals are:

- $\neg AB_a(C)$ represents the abstraction that connections need not be modeled.
- $\neg AB_a(I)$ represents the abstraction that the system is non-intermittent.
- $\neg AB_a(T)$ represents the abstraction that temporal behavior need not be modeled.

Figure 5 shows a portion of the abstraction space for digital circuits along three of the axes of abstraction. This lattice is identical in structure to the ones used in conventional model-based diagnosis for system diagnoses. In conventional model-based diagnosis, each node represents a candidate diagnosis which explains the observations. Each node in Figure 5 represents a candidate meta-diagnosis. The bottom node in the figure represents the meta-diagnosis in which connections, time, and intermittency are not relevant:

$$\neg AB_a(T) \wedge \neg AB_a(C) \wedge \neg AB_a(I).$$

Under the principle that we want to find the simplest meta-diagnosis which explains the observations (and no simpler), we are primarily interested in the minimal diagnoses. For brevity sake, we name meta-diagnoses with the letters corresponding to the abstractions which are AB_a. For example, the meta-diagnosis $\neg AB_a(T) \wedge \neg AB_a(C) \wedge AB_a(I)$ is named by I.

For the example in Figure 2, analysis immediately detects a contradiction and the meta-conflict:

$$AB_a(T) \vee AB_a(C)$$

(this contradiction cannot depend on $AB_a(I)$ as there is only one observation time so far). Figure 6 illustrates the resulting meta-diagnosis lattice. Every meta-diagnosis below the curve is eliminated. The minimal meta-diagnoses are T and C.

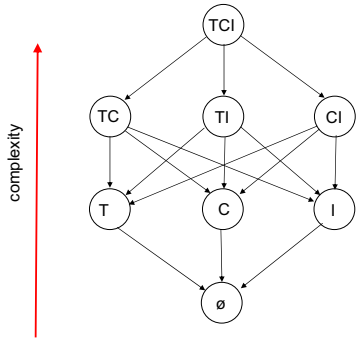


Figure 5: Meta-Diagnosis lattice for digital gates. T indicates temporal models; C indicates connection models; I indicates intermittent models.

5 Modeling components

The conventional MBD model for an inverter is (presuming the usual background axioms define the appropriate functions, domains, and ranges):

$$INVERTER(x) \rightarrow$$

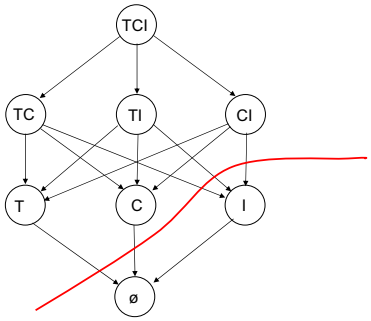


Figure 6: Meta-Diagnosis lattice for digital gates. The meta-conflict $AB_a(T) \vee AB_a(C)$ rules out all meta-diagnoses below the curved line. The minimal meta-diagnoses are T and C.

$$\Big[\neg AB_d(x) \rightarrow [in(x,t) = 0 \equiv out(x,t) = 1]\Big].$$

As this model presumes connections and temporal behavior need not be modeled, in our new framework it is written as:

$$\neg AB_a(T) \wedge \neg AB_a(C) \rightarrow \Big[INVERTER(x) \rightarrow \Big[\neg AB_d(x) \rightarrow [in(x,t) = 0 \equiv out(x,t) = 1]\Big]\Big].$$

Figure 7: Model of inverter under T and C abstractions.

When modeling an inverter as having a delay Δ, the model changes to (labeled T in Figure 5):

$$AB_a(T) \wedge \neg AB_a(C) \rightarrow \Big[INVERTER(x) \rightarrow \Big[\neg AB_d(x) \rightarrow [in(x,t) = 0 \equiv out(x,t+\Delta) = 1]\Big]\Big].$$

5.1 Connection models

To model the inverter to accommodate faults in connections, including bridge faults, requires the introduction of new formalisms. What follows is a brief summary of the formalism presented in [de Kleer, 2007b]. Each terminal of a component is modeled with two variables, one which models how the component is attempting to influence its output (roughly analogous to current), and the other which characterizes the result (roughly analogous to voltage). There are 5, mutually inconsistent, qualitative values for the influence of a component on a node (we refer to these as "drivers").

- $d(-\infty)$ indicates a direct short to ground.
- $d(0)$ pull towards ground (i.e., 0).
- $d(R)$ presents a high (i.e., draws little current) passive resistive load.

- $d(1)$ pull towards power (i.e., 1).
- $d(+\infty)$ indicates a direct short to power.

There are three possible qualitative values for the resulting signal:

- $s(0)$ the result is close enough to ground to be sensed as a digital 0.
- $s(x)$ the result is neither a 0 or 1.
- $s(1)$ the result is close enough to power to be sensed as a digital 1.

Using this formalism produces considerably more detailed component models. We need to expand the $A \equiv B$ in the inverter model to $(A \rightarrow B) \wedge (B \rightarrow A)$. The left half of the inverter model is:

$$\begin{aligned}
&\neg AB_a(T) \wedge AB_a(C) \rightarrow \\
&\quad \Big[INVERTER(x) \rightarrow \\
&\qquad \Big[\neg AB_d(x) \rightarrow \\
&\qquad \Big[[s(in(x,t)) = s(0) \rightarrow d(out(x,t)) = d(1)] \\
&\qquad \wedge [s(in(x,t)) = s(1) \rightarrow d(out(x,t)) = d(0)] \\
&\qquad \wedge d(in(x,t)) = d(R) \\
&\qquad \wedge [d(out(x,t)) = d(0) \vee d(out(x,t)) = d(1)]\Big]\Big].
\end{aligned}$$

We need explicit models to describe how the digital signal at a the node is determined from its drivers. Let $R(v)$ be the resulting signal at node v and $S(v)$ be the collection of drivers of node v. Intuitively, the model for a node is:

- If $d(-\infty) \in S(v)$, then $R(v) = s(0)$.
- If $d(+\infty) \in S(v)$, then $R(v) = s(1)$.
- If $d(0) \in S(v)$, then $R(v) = s(0)$.
- Else, if all drivers are known, and the preceding 3 rules do not apply, then $R(v) = s(1)$.

The resulting model for the node x will depend on $\neg AB_d(x)$ and $AB_a(C)$.

Modeling the inverter to more accurately describe both temporal and causal behavior (labeled TC in Figure 5):

$$\begin{aligned}
&AB_a(T) \wedge AB_a(C) \rightarrow \\
&\quad INVERTER(x) \rightarrow \\
&\qquad \Big[\neg AB_d(x) \rightarrow \\
&\qquad \Big[[s(in(x,t)) = s(0) \rightarrow d(out(x,t+\Delta)) = d(1) \\
&\qquad \wedge [s(in(x,t)) = s(1) \rightarrow d(out(x,t+\Delta)) = d(0)] \\
&\qquad \wedge d(in(x,t)) = d(R) \\
&\qquad \wedge [d(out(x,t+\Delta)) = d(0) \vee d(out(x,t+\Delta)) = d(1)]\Big]\Big].
\end{aligned}$$

The connection models also allow arbitrary bridge faults among circuit nodes. These are described in much more detail in [de Kleer, 2007b].

5.2 Modeling non-intermittency

Figure 8 shows an example where assuming non-intermittency improves diagnostic discrimination. The circuit's inputs and outputs are marked with values observed at times: T_1 and T_2. Note that at T_1, the circuit outputs a correct value and that at T_2, the circuit outputs an incorrect one. By assuming the Or gate behaves non-intermittently, we can establish that the Xor gate is faulty as follows:

If Xor is good, then $\text{In}_1(\text{Xor}, T_1) = 1$. This follows from $\text{In}_2(\text{Xor}, T_1) = 0$, $\text{Out}(\text{Xor}, T_1) = 1$ and the behavior of Xor. Similarly, if Xor is good, then $\text{In}_1(\text{Xor}) = 0$ at T_2. However, if Or behaves non-intermittently, then $\text{In}_1(\text{Xor}, T_2) = 1$. This follows because Or has the same inputs at both T_1 and T_2 and must produce the same output. Thus we have two contradictory predictions for the value of $\text{In}_1(\text{Xor}, T_2)$. Either Xor is faulty or Or is behaving intermittently. Assuming non-intermittency means Xor is faulty.

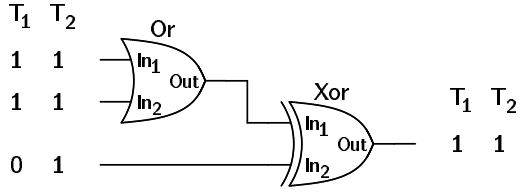


Figure 8: The power of non-intermittency.

All the inferences follow from the defining of non-intermittency:

Definition 13 *[Raiman* et al.*, 1991] A component behaves non-intermittently if its outputs are a function of its inputs.*

This definition sanctions the following inference: if an input vector $\overline{X}$ is applied to an intermittent component at time T, and output Z is observed, then in any other observation T', if $\overline{X}$ is supplied as input, Z will be observed as output.

For the Or-Xor example, the axioms added are:

$$\forall t.\text{Out}(\text{Or}, t) = F(\text{Or}, \text{In}_1(\text{Or}, t), \text{In}_2(\text{Or}, t)) \quad (1)$$

$$\forall t.\text{Out}(\text{Xor}, t) = F(\text{Xor}, \text{In}_1(\text{Xor}, t), \text{In}_2(\text{Xor}, t)) \quad (2)$$

F is a single fixed function for all non-intermittency axioms.

These axioms are implemented in the ATMS/HTMS-based reasoner by deriving prime implicates as follows. At time T_1:

$$AB_d(\text{Xor}) \vee [F(\text{Or}, 1, 1) = 1].$$

At time T_2:

$$AB_d(\text{Xor}) \vee [F(\text{Or}, 1, 1) = 0].$$

Which combine to yield $AB_d(\text{Xor})$.

In the intermittent case, the observation at T_1 equally weights Xor and Or as being correct. If there were other components in the system not affected by the measurement, the observation at T_1 lowers the posterior fault probabilities of Xor and Or.

5.3 Automatic generation of models

The more detailed component models can usually be generated automatically from the most abstract models in a systematic way. In our implementation, the T, C, and I models are automatically derived from the basic $\emptyset$ models by a set of modeling schemas. Consider the most abstract model of an inverter:

$$INVERTER(x) \rightarrow \Big[\neg AB_d(x) \rightarrow [in(x,t) = 0 \equiv out(x,t) = 1]\Big].$$

We use the convention that the function in refers to inputs, and the function out refers to outputs. A non-temporal model can be converted to a simple gate-delay model by replacing every occurrence of $out(x,t)$ (or $out_j(x,t)$) with $out(x, t + \Delta)$.

A non-connection model can be converted to a connection one by first expanding implications, replacing all $in(x,t) = y$ with $s(in(x,t)) = s(y)$ and $d(in,x,t)) = d(R)$ and replacing all $out(x,t) = y$ with $d(out(x,t)) = d(y)$, and adding the usual domain axioms for new variable values.

Non-intermittency requires no change to the component models themselves, but the axioms of Section 5.2 need to be added to the models supplied to the domain MBD.

6 The meta-diagnosis loop

6.1 $\emptyset \rightarrow T$

Consider the three inverter example of Figure 2. The most abstract meta-diagnosis is:

$$\neg AB_a(T) \wedge \neg AB_a(C) \wedge \neg AB_a(I).$$

This meta-diagnosis is supplied to the 'Modeler' module for Figure 4 which chooses the component models at the meta-diagnosis level. The models for all three inverters are described in Figure 7. This produces a failure because all components are known to be good. The 'Meta-Conflicts' module of Figure 4 will construct the meta-conflict:

$$AB_a(T) \vee AB_a(C).$$

$AB_a(I)$ is trivially excluded from the meta-conflict because non-intermittency inferences can only arise when the system has been observed at multiple times.

The abstraction MBD identifies two minimal meta-diagnoses T and C. If both are equally likely, it arbitrarily picks one. Suppose C is chosen. The C models do not resolve the inconsistency either. Figure 9 illustrates the following sequence of inferences with the connection models: (1) Assume the input of A is 1, (2) the causal inverter model drives its output down towards 0, (3) the input of gate B presents a high resistance (low-current) load to its node, (4) the connection model sets the node to 0, (5) the inverter model on B drives its output towards 1, (6) gate C presents a high resistance (low current) load, (7) the connection model sets its node to 1, (8) the inverter drives its output to 0, (9) gate A presents a low resistance (low current) load, and (10) the node model sets the node to 0 which contradicts the input of A being 1. An analogous analysis for the input of A being 0, yields a contradiction as well. The only remaining possible cardinality one diagnosis is T.

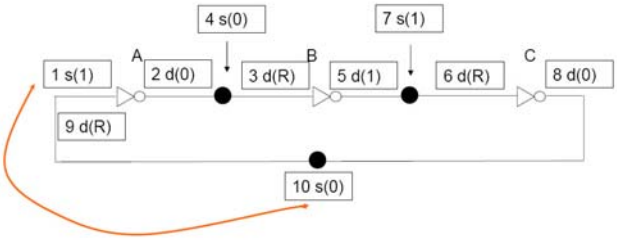


Figure 9: Modeling connections does not remove the failure.

Table 1: Outputs of the inverters of a ring oscillator after t gate delays. The oscillator takes 6 gate delays to return to its initial state, thus the output is a square wave with a period of 6 times the gate delay.

t	0	1	2	3	4	5	6
A	1	1	1	0	0	0	1
B	1	0	0	0	1	1	1
C	0	0	1	1	1	0	0

Using the temporal T models for the inverters produces a consistent analysis demonstrated in Table 1. This circuit is the familiar ring oscillator [Wikipedia, 2007].

6.2 $\emptyset \rightarrow I$

Consider the Or $-$ Xor circuit again (Figure 8). For clarity assume the circuit has one fault. As derived in the Section 5.2, under the $\emptyset$ models, Xor must be faulted. Suppose we measure the output of the Or gate at T_1 and T_2 to be 1 and then 0 respectively. In this case, we have derived the meta-conflict:

$$AB_a(T) \vee AB_a(C) \vee AB_a(I).$$

There are now three minimal candidate meta-diagnoses: T,C,I. The T meta-diagnosis immediately results in a failure yielding the meta-conflict:

$$AB_a(C) \vee AB_a(I).$$

The meta-diagnosis I yields a consistent point of view: Or is failing intermittently. The C meta-diagnosis cannot explain the observation:

$$AB_a(T) \vee AB_a(I).$$

6.3 $\emptyset \rightarrow C$

Consider the Or $-$ Xor example again before the output of the Or gate is observed. Again, for clarity assume the circuit has one fault. Suppose Xor is replaced and the same symptoms reoccur. In this case, both the C and I meta-diagnoses are consistent. Under the I meta-diagnosis, the circuit contains two possible faults:

- Xor is faulted.

- Or is faulted.

The C meta-diagnosis is consistent, with 3 possible faults:

- The node at the output of Xor is shorted to power.
- The connection from the output Xor gate to the node is open and thus it floats to 1.
- The connection to $\text{in}_2(\text{Xor})$ is shorted to ground.

6.4 $\emptyset \rightarrow TCI$

Tasks which require a TCI preferred meta-diagnosis are complicated, but they do occur. Consider the four inverter system of Figure 10. The input to inverter Z is held constant

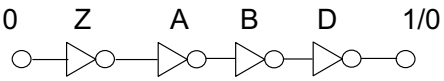


Figure 10: A very simple circuit containing a very hard to pinpoint fault.

at 0. We assume single faults. Observing the output D is usually 0, but outputs 1s with no pattern. The observation $D = 1$ indicates that one of Z, A, B, D is faulted. However, a subsequent observation of $D = 0$ is inconsistent yielding the meta-conflict:

$$AB_a(T) \vee AB_a(C) \vee AB_a(I).$$

No fault in the connections can produce the observations, therefore:

$$AB_a(T) \vee AB_a(I).$$

No temporal fault can lead to this behavior either, so:

$$AB_a(I).$$

Under meta-diagnosis I, the output of A is measured — it is usually 0, but sometimes 1. The output of Z is measured — it is usually 1, but sometimes 0. Therefore Z must be intermittently faulted (under meta-diagnosis I), but replacing it does not change the symptoms. This yields the meta-conflict:

$$AB_a(T) \vee AB_a(C).$$

The CI meta-diagnosis also leads to an inconsistency. There is no fault within the connections that can explain the observations. Likewise there is no fault within the TI meta-diagnosis. The only meta-diagnosis that can explain the symptoms is TCI. The actual fault is an intermittent short between the output of D and output of Z. As the input to Z is 0, its output is 1. The connection models for digital gates are 0-dominant, so that, if a 0 from the output of D were fedback through an intermittent short, it would drive the input to A to 0. Thus for those times in which the intermittent short was manifest, the circuit would be a ring oscillator.

7 Implementation

The analyses described in this paper have been implemented within the ATMS/HTMS framework [de Kleer and Williams, 1987; de Kleer, 1992]. Each domain or abstraction literal is represented by an explicit ATMS assumption in one ATMS instance. A fuller description of the T, C, and I abstractions can be found in [Raiman *et al.*, 1991; de Kleer, 2007b; 2007a; 2007c]. The ATMS/HTMS architecture provides a unified framework to reason over any assumptions, be they about components or abstractions.

8 Related work

Automated model abstraction has a long tradition in Artificial Intelligence. The graph of models ([Addanki *et al.*, 1989]) is similar to the meta-diagnosis lattice (Figure 5) and analyzes conflicts to identify which modeling parameters to change. It is focused on design and analysis and the models that are constructed by hand. It does not use diagnosis to guide the search for models, nor is it applied to diagnosis in some domain. Work on compositional modeling ([Falkenhainer and Forbus, 1991]) also uses ATMS assumptions to represent domain abstractions and conflicts to guide the search for models. Again, the models are constructed by hand and do not use diagnosis at the domain or meta-levels. In context-dependent modeling ([Nayak, 1995]) there is typically a much larger space of model fragments to choose from and explicit context information is used to guide the selection of the domain models. The task is to construct the best causal explanation for a physical phenomena. Yet again, the models are constructed by hand and do not use diagnosis at the domain or meta-levels.

In the model-based diagnosis literature, there has been considerable work on diagnostic assumptions and selecting appropriate models for a diagnostic task [Struss, 1991b; 1991a]. This paper focuses primarily on assumptions associated with choosing domain abstractions.

There has been considerable research on structural abstraction [Chittaro and Ranon, 2004; Hamscher, 1990] where groups of components are combined to form larger systems to reduce computational complexity. [Sachenbacher and Struss, 2005] describes how the task can be used to partition the value of a variable into the qualitative values needed to solve a task. [Torta and Torasso, 2003] presents another approach to partition the value of a variable into qualitative ranges to reduce complexity when there is limited observability of the variables.

9 Conclusions

This paper has presented a general approach to selecting the best domain abstraction level to address a task and has demonstrated it within the context of digital gates. In the case of digital gates the component models can be automatically generated from the basic models using domain schemas.

10 Acknowledgments

Daniel G. Bobrow and Elisabeth de Kleer provided many useful comments.

References

[Addanki *et al.*, 1989] S. Addanki, R. Cremonini, and J. S. Penberthy. Reasoning about assumptions in graphs of models. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence*, pages 1432–1438, Detroit, MI, August 1989.

[Böttcher *et al.*, 1996] C. Böttcher, P. Dague, and P. Taillibert. Hidden interactions in analog circuits. In Suhayya Abu-Hakima, editor, *Working Papers of the Seventh International Workshop on Principles of Diagnosis*, pages 36–43. Val Morin, Quebec, Canada, October 1996.

[Böttcher, 1995] Claudia Böttcher. No faults in structure? how to diagnose hidden interactions. In *IJCAI*, pages 1728–1735, 1995.

[Chittaro and Ranon, 2004] Luca Chittaro and Roberto Ranon. Hierarchical model-based diagnosis based on structural abstraction. *Artif. Intell.*, 155(1-2):147–182, 2004.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, April 1987. Also in: *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufmann, 1987), 280–297.

[de Kleer *et al.*, 1992] J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[de Kleer, 1992] J. de Kleer. A hybrid truth maintenance system. PARC Technical Report, January 1992.

[de Kleer, 2007a] J. de Kleer. Diagnosing intermittent faults. In *18th International Workshop on Principles of Diagnosis*, Nashville, USA, 2007.

[de Kleer, 2007b] J. de Kleer. Modeling when connections are the problem. In *Proc 20th IJCAI*, pages 311–317, Hyderabad, India, 2007.

[de Kleer, 2007c] J. de Kleer. Troubleshooting temporal behavior in "combinational" circuits. In *18th International Workshop on Principles of Diagnosis*, Nashville, USA, 2007.

[Falkenhainer and Forbus, 1991] B. Falkenhainer and K. D. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51(1-3):95–143, 1991. Also in: J. de Kleer and B. Williams (eds.) *Qualitative Reasoning about Physical Systems II* (North-Holland, Amsterdam, 1991 / MIT Press, Cambridge, Mass., 1992).

[Hamscher, 1990] W. C. Hamscher. XDE: Diagnosing devices with hierarchic structure and known component failure modes. In *Proc. 6th IEEE Conf. on A.I. Applications*, pages 48–54, Santa Barbara, CA, March 1990.

[Nayak, 1995] P. Pandurang Nayak. *Automated Modeling of Physical Systems*. Springer Verlag LNCS 1003, Cambridge, MA, 1995.

[Raiman *et al.*, 1991] O. Raiman, J. de Kleer, V. Saraswat, and M. H. Shirley. Characterizing non-intermittent faults. In *Proc. 9th National Conf. on Artificial Intelligence*, pages 849–854, Anaheim, CA, July 1991.

[Sachenbacher and Struss, 2005] Martin Sachenbacher and Peter Struss. Task-dependent qualitative domain abstraction. *Artif. Intell.*, 162(1-2):121–143, 2005.

[Struss, 1991a] P. Struss. A theory of model simplification and abstraction for diagnosis. In *Proc. 5th Int. Workshop on Qualitative Physics*, Austin, TX, May 1991.

[Struss, 1991b] P. Struss. What's in SD? Towards a theory of modeling for diagnosis. In L. Console, editor, *Working Notes of the 2st Int. Workshop on Principles of Diagnosis*, pages 41–51. Technical Report RT/DI/91-10-7, Dipartimento di Informatica, Universitá di Torino, Torino, Italy, October 1991.

[Torta and Torasso, 2003] Gianluca Torta and Pietro Torasso. Automatic abstraction in component-based diagnosis driven by system observability. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 394–402. Morgan Kaufmann, 2003.

[Wikipedia, 2007] Wikipedia. Ring oscillator — Wikipedia, the free encyclopedia, 2007. [Online; accessed 12-February-2007].

Operating part model for on-line diagnosis

Eric Deschamps*, Sébastien Henry and Eric Zamaï***

*Laboratoire des Sciences pour la Conception, l'Optimisation et la Production (G-SCOP),
INPGrenoble, UJF, CNRS,
46 avenue Félix Viallet, 38031 Grenoble Cedex, France
eric.deschamps@inpg.fr, eric.zamai@inpg.fr

**Université de Lyon, Lyon, F-69003, France ; université Lyon 1, Laboratoire d'Informatique
pour l'Entreprise et les Systèmes de Production (LIESP), Villeurbanne, F-69622, France.
sebastien.henry@iutb.univ-lyon1.fr

Abstract

The objective of the Automated Manufacturing System reconfiguration is to react to failures such as breakdowns of actuators or sensors. Following such events, the control laws run by the control system are often blocked and thus become inadequate. Thus, it is therefore necessary to reconfigure the control system. This reconfiguration process is inevitably based on the capacities still offered by the operating part to achieve the production objectives. In this context, the paper proposes an on-line function diagnosis able to provide the required information on the capacities of the operating part. It is based on a model of the operating part in normal operation, and generic rules to obtain the possible origins and consequences of a detected symptom of failure.

Introduction

Automatic Manufacturing Systems (AMS) are composed of two different parts: the controlled system and the control system (as shown in Fig. 1). The controlled system is composed of actuators which act on the product flow. It is also composed of sensors to control and monitor the actuators and the product flow although this observability on these components is only partial. To manage the complexity of such controlled systems, a hierarchical and modular control is often proposed (John, 1989). Each elementary part of the controlled system is controlled by its local control module, and these two elements are called Functional Chain (FC) as shown in Fig. 1. Thus, the purpose of the coordination level is to manage a set of FCs by using services offered by these FCs. Throughout these levels, a high level request is broken down into a sequence of elementary requests until level 1 is reached. At a considered level i, when a request is executed before the due date, sent with the request, an execution report is generated and sent to level i+1. However, at level 1, in the event of breakdown of actuators or sensors, the due date cannot always be respected and a faulty execution report is sent. On reception of such a report, level 2 tries to confine the failure before the due date of the current request to avoid sending a faulty execution report to level 2.

To ensure that the operation takes place as defined above, each module must integrate supervision, monitoring and control (SM&C) functions such as detection, diagnosis, prognosis and decision (Combacau, et al., 1990). Thus the operation of the control system is based first on collaboration between these functions and second on the exploitation of models. Here, the paper focuses on the function diagnosis. The literature proposes two kinds of approaches: first, model-free methods which are based on empirical knowledge of the controlled system as presented in (Bohez, et al. 1997); and second, model model-based methods as presented in (Ressencourt et al. 2006). The paper deals with a method for an on-line diagnosis using a model of the operating part; it is based on other classical approaches of diagnosis with an added original approach which is presented in the next sections.

This paper is organized as follows. First, section 2 focuses on the possible objectives of diagnosis. Based on the objective on which the paper focuses, section 3 presents the principle of the diagnosis. Section 4 presents an academic example, and the corresponding model is presented in section 5. Finally, section 6 details the diagnostic approach.

Objective of the diagnosis

In the reactive hierarchical control, the diagnosis has two main objectives: first to identify the component responsible for the detected symptom with the aim of repairing it as fast as possible. The second objective is to provide information on the availability of services, to use this to confine the failure and execute the current request before the due date. The proposed diagnosis in this paper gives a solution for attaining the second objective at the coordination level. Thus, it provides information on the availability of FC services.

At the coordination level, to react to a hypothetical faulty execution report sent by a FC, the control system must be reconfigured. This reconfiguration consists in designing automatically a new suitable control law (set of requests to FCs). However, to take such a decision, it is

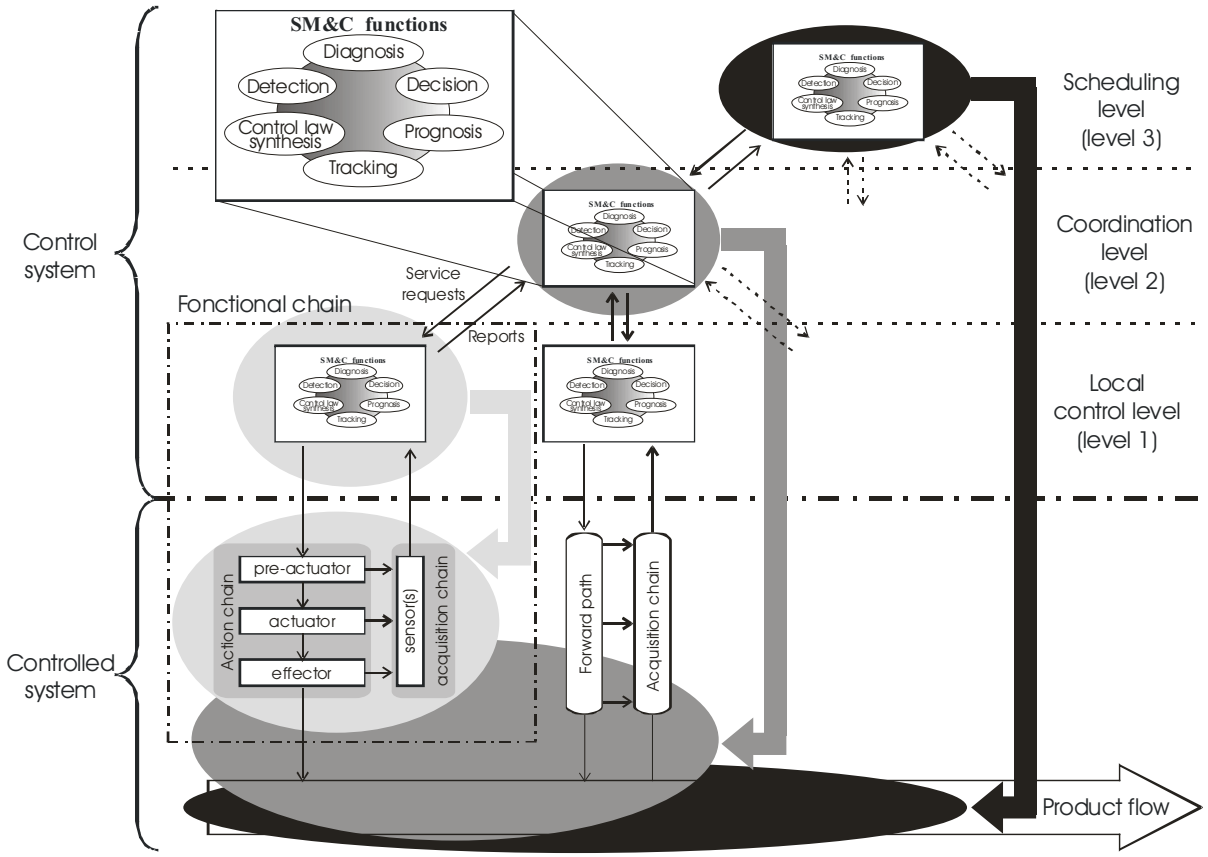


Fig. 1. Diagram of an AMS

necessary to have a realistic view of the services still offered by FCs even if this view is pessimistic considering the time to react, fixed by the due date. Thus several questions must be asked. What model(s) must be used by such a function diagnosis? The objective being to reconfigure, how to link the diagnosis to the reconfiguration? And finally, how can this pessimistic view be expressed?

General situation

In discrete event systems, methods for diagnosis are classically based on tools such as state machines (Sampath, et al., 1998), or Petri nets (Genc, et al., 2003). From detection and the partial observation of the controlled system, these methods try to identify in which state the controlled system is and what the origin of the faulty execution report is. In this kind of approach, it is generally proposed to model all the possible failures. However, due to the resulting complexity at the coordination level, it would be very difficult to predict all the failures. So, in the event of unpredicted failure, the diagnosis cannot provide a result. At this level of control, to guarantee the reactivity of the control system, the diagnosis must not be based only on a failure model.

After a failure occurrence, a method is also proposed in (de Jonge et al., 2006) to make a diagnosis based on two observations of the system which is the failed action of a plan. The method is based on the theory of diagnosis from first principles (Reiter, 1987). In fact, plan execution can be diagnosed by viewing action instances of a plan as components of a system and by viewing the input and output objects of an action as input and outputs of a component. This makes it possible to apply classical model-based diagnosis to plan execution. This idea can be exploited in the context of our work by viewing an executed service as an action. However, this approach must be extended. First, in (de Jonge et al., 2006), actions are considered as independent. This hypothesis cannot be used; several services can be carried out by the same functional chain, and so are not independent. Second, the inputs of components (corresponding to pre-conditions of action or pre-conditions, conditions, pre-constraints and constraints of services as described bellow) can have an impact on the behavior of the services but also on the operation of the functional chain. The last problem is the size of the model. In fact, the diagnosis from first principles considers a finite set of components or the plan diagnosis a finite set of action. However, due to the cyclical operation of the control system at the coordination

level, the number of the past executed services is not finite. It is therefore necessary to reduce the model used by the diagnosis under hypotheses which will be explained next.

In this paper, the model for diagnosis is based on the formal description of services behavior proposed in (Henry, et al., 2004). This formal description can be used to design control laws but also to provide information on the behavior of services offered by FCs for diagnosis as explained in section 6. Thus, to reconfigure the control system after the diagnostic, it is necessary to express the diagnostic result onto the formal description proposed in (Henry, et al., 2004). Information on services is not sufficient for reconfiguring the control system. In fact, information on the state of the controlled system is also needed (Henry et al., 2003). More precisely, the diagnostic method provides on a rating scale a qualification of the availability of services offered by FCs. In this way the vocabulary from the safety field of research will be used. A service will be qualified as:

- Correct: observed correct (direct/indirect) in its last use.
- Not suspected: not observed but not suspected for the faulty execution report.
- Suspect: its last use could be the possible origin of the faulty execution report, or its availability could be affected by the initial origin of the faulty execution report.
- Lost: confirmed unavailable (another diagnostic result for instance).

The function diagnosis will qualify variable states which characterize the state of the FCs and of the product flow as:

- Correct: observed correct (direct/indirect).
- Not suspected: not observed but not suspected for the faulty execution report.
- Suspect: can be affected by the initial origin of the faulty execution report.
- Incorrect: affected by the initial origin.

After a presentation of an example which illustrates the rest of the paper, the model used by the function diagnosis and the method for diagnosis based on the preceding rating scale are presented.

Academic example

The example used to illustrate the proposed function diagnosis is composed of two cylinders (as shown in Fig. 2). The first allows the transfer of a product from position A to position B, and the second, from position C to position D. A conveyor, which is always in operation mode, transfers the product from position B to C and from C to D. Three sensors are added to detect a product in positions A, D and E. Thus the coordination levels manage six FCs, corresponding to each element which makes up the operating part. In the rest of the paper, only products which must be put in D are considered. The next part provides a description of services behavior.

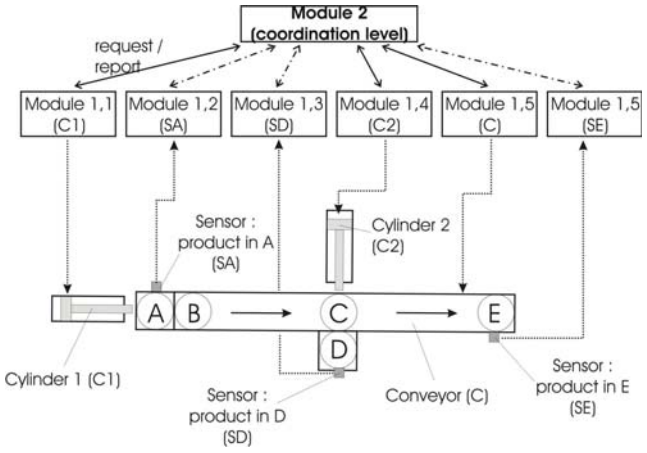


Fig. 2. Example

Description of service behavior

The description of services behavior is based on the operation concept (Ghallab, et al., 2004) which was extended for the control law synthesis of Discrete Event Systems in (Henry, et al., 2004). Finally, in the framework of monitoring and diagnosis, this description has been extended in (Deschamps, et al., 2005). The main important points of this description are presented below.

Services description

From the coordination point of view, an operation specifies (see Fig. 3):

- one service offered by an FC,
- the effects of this service on the operating part,
- the possible effects on the product flow,
- the pre-conditions to obtain these possible effects,
- the conditions to continue these effects,
- the pre-constraints which must be true before operation,
- the constraints which must be true during the operation.

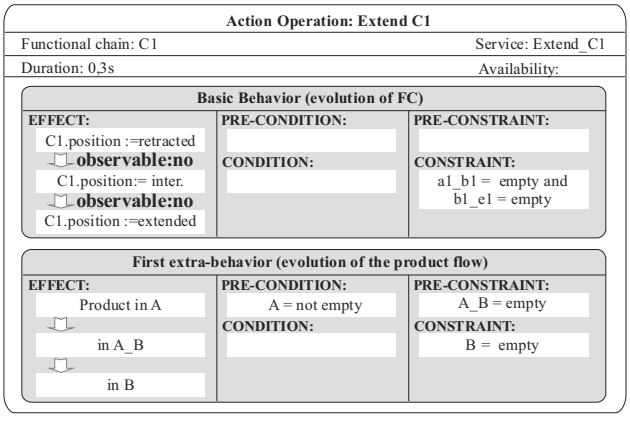


Fig. 3. Operation model of *extend cylinder 1*

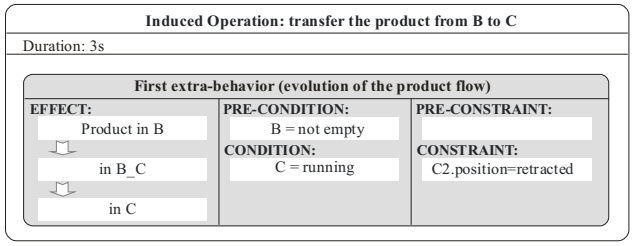


Fig. 4. Example of induced operation

From one state of the operating part and the product flow, the execution of a service involves the evolutions of the corresponding FC and of the product flow whose pre-conditions and conditions (noted (pre-)conditions) are true from this state. Moreover, all the (pre-)constraints of every evolution whose pre-conditions are true must be true. It should be noted that the model does not represent explicitly the set of all the possible states of the controlled system. This not only makes the modeling step more accessible for an expert, but it also avoids the problem of combination explosion.

As explained in section 1, FCs contain SM&C functionalities. Thus, if the observability which they have on the controlled sub-system is sufficient, they can contain an efficient detection process. As explain bellow, this information is essential to the coordination level in order to limit the search for the diagnosis. As shown in Fig. 3, this information is provided to the coordination level by the data field "observable: yes/no". Finally, an operation can be uncontrollable, for example the operation *transfer a product from B to C* presented in Fig. 4. This particular operation is called induced operation, because the corresponding evolution is carried out only if the (pre-)conditions are true.

Monitoring services and detection

In the previous section, the information given by the observability of FCs on the operating part has been presented. In the same way, information given by monitoring sensors on the product flow (sensor of SA, SD, SE in the example) are also needed to determine the reliability of the information on the state of the controlled system. In fact, some sensors on the product flow are used in AMS only for monitoring purposes. These monitoring

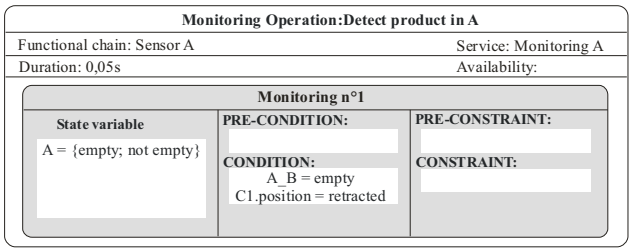


Fig. 5. Example of monitoring operation

sensors allow the verification of the values of product state variables. Thus, monitoring operations are introduced (as shown in Fig. 5). Such operations define the set of state variables and the corresponding values which can be verified. As in all generic operation descriptions, There may be exist (pre-)conditions and (pre-)constraints. These operations are inserted in the controlled law by the synthesis algorithm (Henry, et al., 2004), in order to give to FCs, the knowledge on the product flow behaviors (as shown in Fig. 6). Therefore, a detection function can be implemented in these particular FCs as in all FCs which contain first an acquisition chain, and second, a model of the temporal behavior of the action chain. Following a service execution, if the information provided by these two elements is not coherent, the functional chain sends to the coordination a faulty execution report. This mechanism monitors the services executions. If the coordination level does not receive a faulty execution report from these FCs, it can conclude that the corresponding services execution is reliable. It should be noted, that the function diagnosis presented which is located in the coordination level, is launched by this faulty execution report.

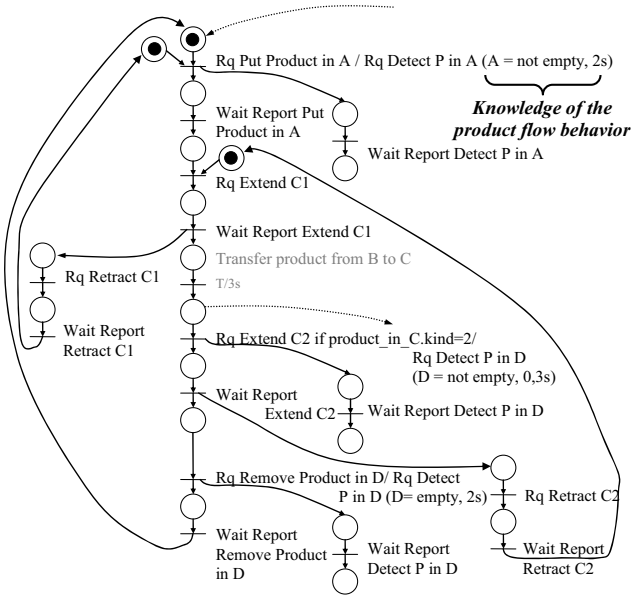


Fig. 6. Control law for the coordination level

Function diagnosis

The proposed function diagnosis consists in searching in the past evolution of the controlled system, for what the possible origins (past executed services) of a faulty execution report could be. Moreover, from the possible origins, it searches for the possible other consequences on the services availability and on the controlled system state. From this result, it qualifies services and the state variables of the controlled system by using the rating scales previously proposed. The proposed function diagnosis is

presented in three points: first, a model generated in normal operation (before a faulty execution report) which will give the diagnosis all the necessary information, second, a mechanism based on the exploitation of the controlled system observability in order to reduce this model used by the diagnosis. Finally, the method for diagnosis is presented.

Diagnosis model

The aim of the diagnosis is to know, firstly, the possible origins which can lead to the faulty execution report and, secondly, what the consequences are on the availability of services and on the state variables of the controlled system. A faulty execution report may correspond, first, to a failure of the FC which executes the service and, second, to a failure propagation process through the operating part (Chang, et al., 1991). In fact, even if an FC is operational, if the controlled system is not in a state compatible with the launching of one of its services, the effect of the requested service will not be the expected effect. Thus a failure can be propagated through the operating part by the non-respect of (pre-)constraints and (pre-)conditions which should have been respected. The consequences of the non-respect of (pre-)constraints and (pre-)conditions are the partial execution or the non-execution of the corresponding effect and perhaps damage to the FC which launches the service. All this information can be obtained from the description of the services and from the control law. In fact, the control law contains all the previously requested services, and the description of the services contains all the (pre-)constraints, (pre-)conditions, effects of these services. In a real-time context, it is important to make as rapid a diagnosis as possible. Our approach therefore proposes generating, during normal operation, a diagnosis model which contains all of the necessary information previously described. A illustration of this information, corresponding to a particular service is given in Fig. 7. Each time that a request is sent and a report is received by the coordination level, information (behavioral and structural) corresponding to this graphic illustration is added to the model. For the academic example, Fig. 8 represents the model during the execution of the operation *extend cylinder 2* (EC2). It is composed for the past executed services (defining the behavior of the model), all the (pre-)constraints, (pre-)conditions and effects (which define the structure of the model, i.e. the links between services). To avoid the size explosion of the model, it is next reduced to information which is not qualified as correct with the exploitation of the observability.

Exploitation of the observability

The observability of the controlled system is used to limit the search for of the diagnosis. To do this, let us define the following hypothesis: "if a requested evolution of the controlled system is correlated with a change in state of a sensor, then this evolution is correct". In fact, the probability of a failure occurring in a sensor which

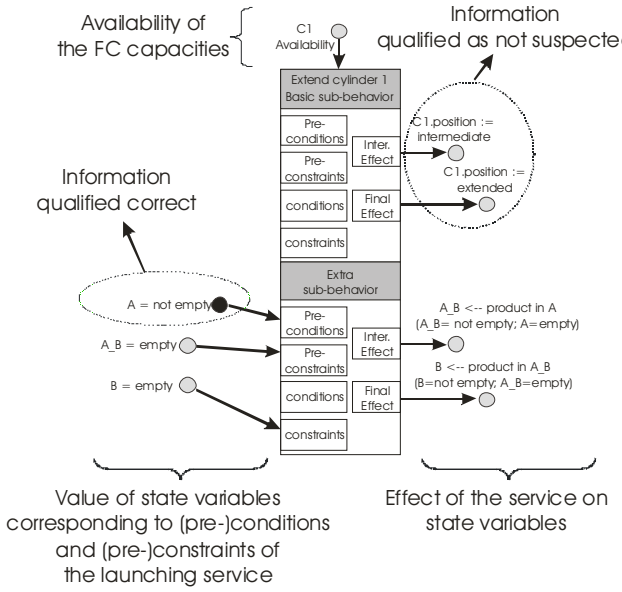


Fig. 7. Recorded information for a service

observes the end (or the beginning) of an operation, at the same time as the end (or the launching) of the corresponding operation is extremely low. This hypothesis makes it possible to eliminate partly some of the possible failures. Even if the coordination level is not directly connected to sensors, it can have this information through the data field observability of services or through the launching of the monitoring operation. For instance, following the service *put product in A*, the FC SA does not send to the coordination level a faulty execution report following the request *detect Product in A*. This implies that this FC has made a correlation between the knowledge concerning the behavior of the product sent by the coordination level (see Fig. 6), and the observation of this product. And thus the coordination level qualifies the execution of this service and the information product in A as correct. This corresponds to zone B in Fig. 8 which is removed from the model, and the node Product in A colored in black (as shown in Fig. 7).

Moreover, the reduction method considers that if the execution of a service is qualified as correct, then the corresponding (pre-)constraints and (pre-)conditions can also be qualified as correct. For instance, in Fig. 8 zone C was removed from the model following the service EC2 of the previous operating cycle. In fact, the execution of EC2 and the information *product in D* is qualified as correct following EC2 execution as well as corresponding (pre-)constraints and (pre-)conditions.

From this reduction, the search for the possible origins of a faulty execution report or the other consequences is limited to past evolutions of the controlled system which are not qualified as correct according to the reduced model. In fact, if the origin (or the consequence) of a faulty execution report is observable, it should have been previously detected. The next part presents the method to

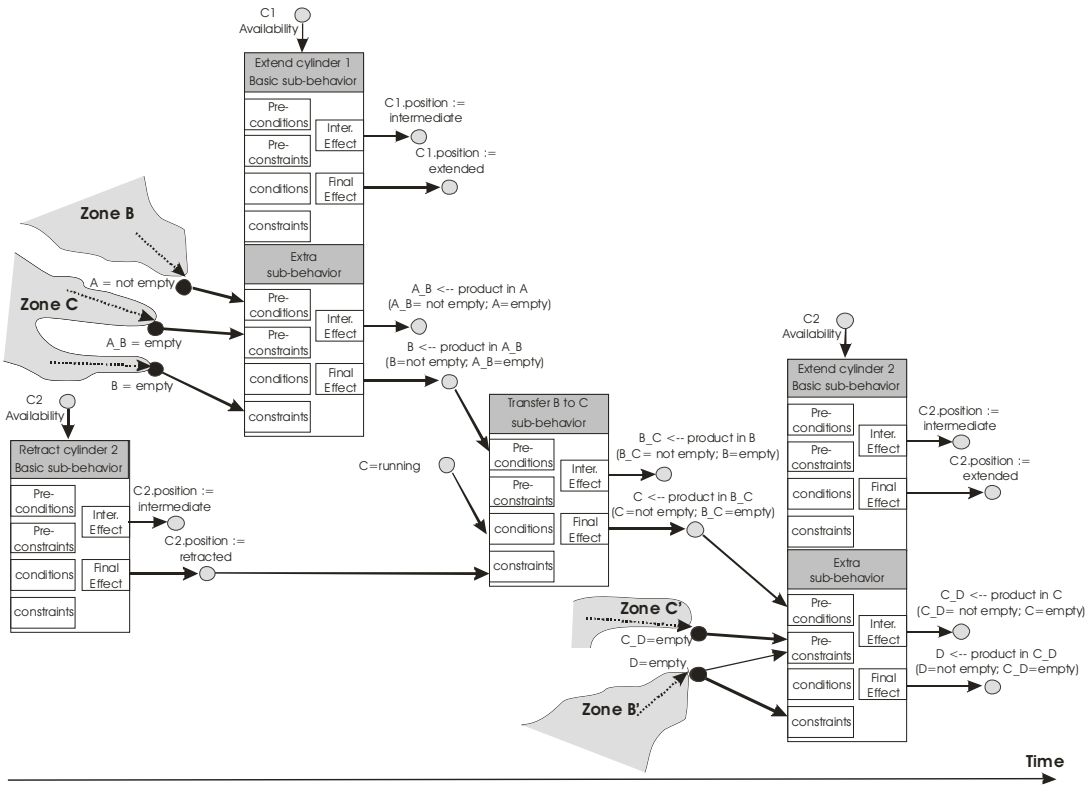


Fig. 8. Model in normal running

qualify as suspect or not suspect first the services offered by the controlled system and second the state variables of the controlled system.

Diagnosis principle

The model previously presented is completed during the dynamical operation of the controlled system and provides all the needed information for the diagnosis which is launched at the coordination level after the reception of a faulty execution report. The aim of the function diagnosis is to propagate the doubt in this model. This doubt corresponds to information which must be qualified as suspect. The propagation uses rules based on the interpretation of the description of services. As this description represents the normal operation of the controlled system, the rules obtained are necessarily pessimistic. They take the worst case into account. In fact, the "propagation of doubt" must not forget to suspect services which are the possible origins of the faulty execution report and the other possible consequences on availability of other services and on the controlled system state. For example, if a constraint was not respected during a service execution, it is possible that the corresponding FC was not damaged. However, the description of services contains only information on normal operation and thus this conclusion cannot be obtained from such a description. The rules of the propagation to obtain, first, the possible origins and, second, the possible other consequences, are

propagation rules					
basic behavior		**effect of basic beh.**	**effect of extra-beh.**	**availability of F.C.**	**condition**
F.C. availability suspect		suspect	suspect	suspect	-
F.C. availability lost		suspect	suspect	lost	-
pre-cdt	**suspect**	suspect	suspect	-	-
pre-cdt	**incorrect**	incorrect	incorrect	-	-
cdt	**suspect**	suspect	suspect	-	-
cdt	**incorrect**	incorrect	incorrect	-	-
pre-ctrt	**suspect**	suspect	suspect	suspect	if effect of
pre-ctrt	**incorrect**	suspect	suspect	suspect	extra-
ctrte	**suspect**	suspect	suspect	suspect	behavior i
ctrte	**incorrect**	suspect	suspect	suspect	not incorrect
extra-behavior i		**effect of extra-beh. i**	**effect of other beh.**	**availability F.C**	**condition**
pre-cdt	**suspect**	suspect	-	-	-
pre-cdt	**incorrect**	incorrect	-	-	-
cdt	**suspect**	suspect	-	-	-
cdt	**incorrect**	incorrect	-	-	-
pre-ctrt	**suspect**	suspect	suspect	suspect	-
pre-ctrt	**incorrect**	suspect	suspect	suspect	-
ctrt	**suspect**	suspect	suspect	suspect	-
ctrt	**incorrect**	suspect	suspect	suspect	-
unexpected extra-behavior j		**no effect of extra-beh. j**	**effect of other beh.**	**availability F.C**	**condition**
pre-cdt nv.	**suspect**	suspect	-	-	-
pre-cdt nv.	**incorrect**	incorrect	-	-	-
pre-cdt v.	**suspect**	suspect	-	-	-
cdt nv.	**suspect**	suspect	-	-	-
cdt nv.	**incorrect**	incorrect	-	-	-
cdt v.	**suspect**	suspect	-	-	-
pre-ctrt nv.		suspect	suspect	suspect	^
pre-ctrt nv.	**suspect**	suspect	suspect	suspect	I
pre-ctrt v.	**suspect**	suspect	suspect	suspect	if no effect of
pre-ctrt v.	**incorrect**	suspect	suspect	suspect	extra-
ctrt nv.		suspect	suspect	suspect	behavior j
ctrt nv.	**suspect**	suspect	suspect	suspect	incorrect
ctrt v.	**suspect**	suspect	suspect	suspect	I
ctrt v.	**incorrect**	suspect	suspect	suspect	v

Fig. 9. Propagation rules

based on the table presented in Fig. 9. Several points should be noted:

• The propagation to obtain the origins of incorrect information (following verification by observability of the product flow as explain bellow) or suspect information is different. As shown in Fig. 9, if a pre-condition (noted pre-cdt) is suspect, the corresponding effect (intermediate and final effect) is suspect. However, if the pre-condition is incorrect the corresponding effect is incorrect (not executed).

• The propagation is different for basic behaviors and for extra-behaviors. For example, if the effect of a basic behavior is suspect, all other effects are suspect. However, if the effect of an extra-behavior is suspect, only this extra-behavior is suspect.

• The rules for an unexpected extra-behavior (an effect on the product which is not expected to occur without failure ensuing) are more complex. In fact, unlike expected extra-behavior all the (pre-)constraints are not necessarily verified because this behavior is unexpected. Moreover, all the (pre-)conditions are not necessarily not-verified (nv), and thus must be distinguished in the rules.

• If a service has violated a (pre-)constraint, all the services which are then executed by the same FC are qualified as suspect (as shown in the rule by the column availability of FC). In the same way, if the origins of the unexpected behavior of a service could be the FC which launches this service, all services all the services which are then executed by the same FC are also qualified as suspect.

• The fifth column corresponds to the condition needed to apply the rule of the corresponding line. For example, if the behavior are not executed (i.e. its effect is incorrect) even if pre-constraints or constraints is not true (i.e. suspect or incorrect), the availability of FC is not suspected. In fact, no pre-constraint or constraint is violated.

The principle used for the diagnosis is the one called values propagation through the data structure, by application of these rules. The problem here corresponds to the problem of obtaining minimal conflicts in the diagnosis from first principles. It is not necessary to obtain the diagnoses. In fact, the proposed diagnosis method must provide only a set of suspect services and the suspect state variables. To do this, first, all the possible origins of the first suspect information (coming from the report of faulty execution) are searched for and, second, all the other possible consequences. During this search, the possible paths of the failure propagation, given by the rules, are recorded in the model as shown in Fig. 10. In fact, the observability of the product flow can validate or invalidate the path of the failure propagation. In the example, if cylinder 2 is blocked in position retracted, the product which must be put in D goes in E. The information given by the FC SE on the presence of the product in E can validate this hypothesis. Thus, the information noted S1, S1.O2.C1, S1.O2.C2, S1.O2.C3, S1.O2.C3.C1, S1.O2.C3.C1 can be qualified as incorrect and S1.O2 as lost, because the only possible origin of the two observations is S1O2. More generally if a path of failure propagation is validated, the information corresponding to this path is qualified as incorrect (or lost for the availability

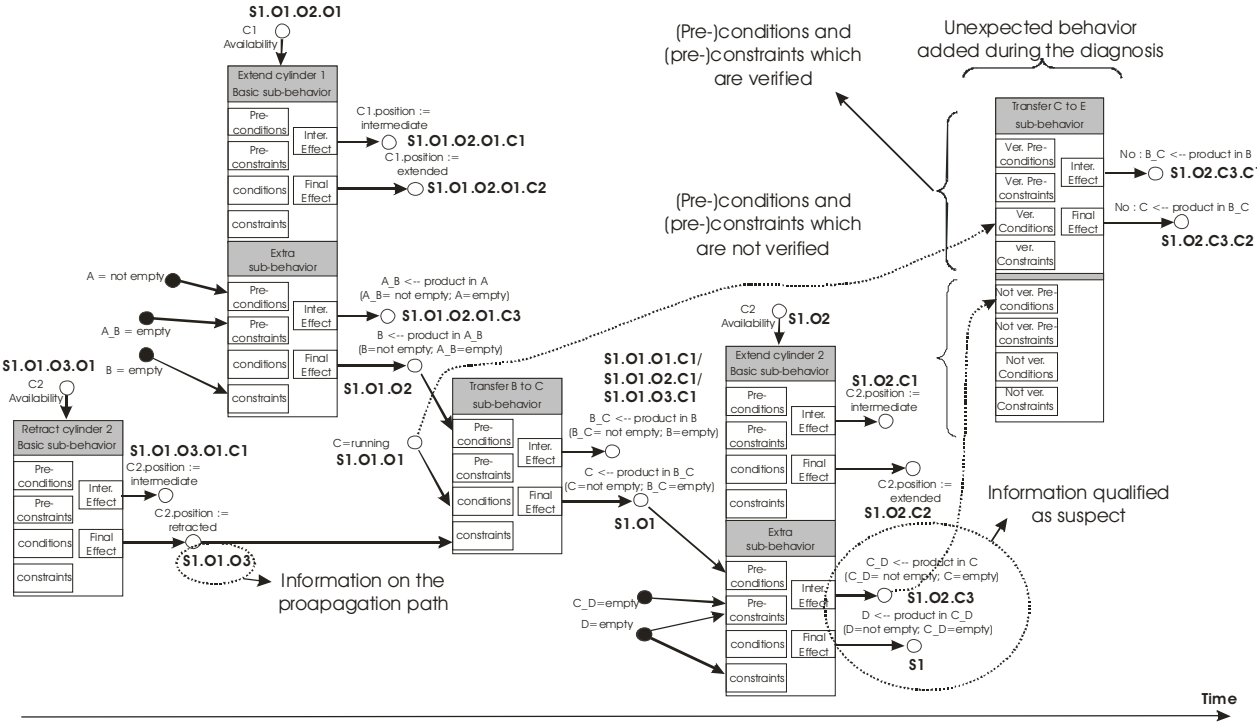


Fig. 10. Failure propagation paths

of services). From this new information qualified as incorrect, a new search for origins and consequences is performed.

Thus the model contains all the suspect and incorrect information. This information must be projected on the description of services and on the state of the controlled system so that it can be exploited in the reconfiguration context. This projection is based on simple rules like:

• All services of an FC whose availability is qualified as suspect in the data structure are qualified as suspect (in data field availability as shown in Fig. 3).
• All services of an FC whose availability is qualified as lost in the data structure are qualified as incorrect .
• All state variables corresponding to a state variable which is qualified as suspect at its last appearance in the data structure are qualified as suspect.
• All state variables corresponding to a state variable which is qualified as incorrect at its last appearance in the data structure are qualified as incorrect.

Thus, the function decision must be able to use flexibilities still offered by the operating part (i.e. services which are not qualified as lost or suspect) to design a new appropriate control law, by taking error and doubt in the state of the controlled system into account.

Conclusion and future work

This paper deals with a function diagnosis which qualifies the services of a functional chain and the state variables of the controlled system. The diagnosis is based, first, on a model which contains the past evolutions of the controlled system. This model is limited to information which is not qualified as correct in order to limit its size. Secondly, from the description of services in normal operation, rules concerning the abnormal operation can be deduced. When a faulty execution report appears, this model and these rules can help determine which services, along with other possible consequences on availability of other services and on the controlled system state, are the possible causes of this situation. They are then able to be qualified as suspect, incorrect or lost. This qualification, based on generic rules as expert systems gives an efficient function diagnosis for the computational time without needing to model all the possible abnormal situations. Finally, the projection of the suspect and incorrect information on the model is carried out to reconfigure the control system.

Our future work will first focus on the search for other hypotheses to decrease the size of the data structure. The rules for abnormal operation must be formalized, and finally, the principle of the diagnosis must be implemented and tested on a real case. Moreover, the approach must take into account possible results of diagnosis sent by FCs so that the diagnosis can be refined at the coordination level (impact of the identify origins on the other FCs). In fact, when the first origin is known the consequences should be known, from the model and the rules for abnormal operation. Such a result should lead to a precise update of the services description.

References

Bohez, E.L.J. ; Thieravarut, M. 1997. Expert system for diagnosing computer numerically controlled machines: a case-study. *Computers in Industry* 32(3): 233-248.

Chang, S. J.; DiCesare, F. and Goldbogen, G. 1991. Failure Propagation Trees for Diagnosis in Manufacturing Systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21(4):767-776.

Combacau, M.; Berruet, P.; Zamai, E.; Charbonnaud, F. and Khatab, A. 2000. Supervision and monitoring of production systems. In *Proceedings of the IFAC Conference on Management and Control of Production and Logistics.*

de Jonge, F.; Roos, N. and Witteven, C. 2006. Primary and secondary plan diagnosis. In *Proceedings of the International Workshop on Principles of Diagnosis (DX'06).*

Genc, S.; Lafortune, S. 2003. Distributed diagnosis of discrete-event systems using Petri nets. In *Proceedings of the International Conference on Application and Theory of Petri Nets*.

Deschamps, E.; Henry, S. and Zamaï, E. 2006. Synchronization of Operating Part Model in Failure Context. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation.*

Ghallab, M.; Nau, D. and Traverso, P. 2004. *Automated planning, Theory and Practice*, Elsevier.

Henry, S.; Deschamps, E.; Zamaï, E. and Jacomino, M. 2004. Control law synthesis algorithm for discrete-event systems. In *Proceedings of the IFAC Conf. on Management and Control of Production and Logistics*.

Henry, S.; Zamaï, E. and Jacomino, M. 2003. Decisional Requirements for Supervision, Monitoring and Control Structures. In *Proceedings of the IMACS Multiconference. Computational Engineering in Systems Applications*.

Jones, A. 1989. A Multi-layer / Multi-level Control Architecture for Computer Integrated Manufacturing Systems. In *Proceedings of the IEEE International Symposium on Intelligent Control*.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96.

Ressencourt, H.; Travé-Massuyès, L. and Thomas, J. 2006 Hierarchical modelling and diagnosis for embedded systems. In *Proceedings of the International Workshop on Principles of Diagnosis (DX'06)*.

Sampath, M.; Lafortune, S. and Teneketzis, D. 1998. Active diagnosis of discrete-event systems. *IEEE transactions on automatic control* 43(7):908-929.

Fault Tolerant Estimation with Sensor Redundancy Management in Distributed Dynamical Systems by Means of Nonlinear Federated Filtering*

A. Edelmayer,† M. Miranda,‡ L. Nádai

Systems and Control Laboratory
Computer and Automation Research Institute of Hungarian Academy of Sciences
Kende u. 13-17, Budapest, H-1111, Hungary

Abstract

This paper discusses the application of the idea of federated filtering to state estimation of intrinsically nonlinear distributed systems by examining its impacts on filtering performance by using the Extended Kalman Filter (EKF) as state estimator. Specifically, the performance of the traditional centralized solution is compared with the filtering structure obtained using the federating idea, and their conceptions, and their ability to balance between fault tolerance and estimation accuracy is examined. Our research demonstrates how successfully the EKF for solving nonlinear estimation problems in federated structures can be used, noting that the idea of federation have only been demonstrated for linear problems previously. In addressing the demands of both fault tolerance and estimation accuracy, it is shown that increased filtering accuracy is relied on the proper choice of sharing of the error covariance information between local filters, while sensor fault tolerance is provided by the utilization of an appropriate resetting policy of the filter.

Introduction

Distributed systems are increasingly important in a widening array of applications in process control, information and communication systems, sensor networks and vehicle technology. The decentralized implementations do not just benefit from a systems design approach by providing more cost-effective solutions, but they are required for effective functioning. Advanced vehicle onboard control systems, including land, marine and avionic systems, for instance, are increasingly relied on a highly distributed electronic systems architecture. These architectures might contain a dozen of subsystems decomposing overall system functionality to several subsystem functions, which are then individually controlled and supervised by one or more dedicated Electronic Control Units (ECU's). The multitude of these ECU's are typically implemented on embedded platforms and interconnected via heterogeneous computer networks. A fundamental problem in such kind of distributed control structures is to solve detection and estimation problems to supply reliable data for controllers, using scalable algorithms which comply with the stringent performance requirements (*cf.* real-time execution, complexity, modularity, computational energy, etc.) demanded by embedded applications.

As a matter of fact, the dynamics of these systems are characteristically nonlinear, moreover, the representative application domains of this technology tend to be, more and more frequently, safety critical. This requires development of novel fault tolerant methods and algorithms properly fitting to the distributed character of the application problem, for estimation, and particularly fault detection, that are currently unavailable.

It has obvious advantages in assigning the overall computation burden among local filters, and in fault tolerant capability in large-scale distributed systems. The centralized filter suffers from the size (dimension) of the estimation problem and is prone to both sensor and implementation failures and the solution cannot answer the calls of the new distributed architectures. The alternative is to process the information locally at the component or subsystems level using local information about the system dynamics and use a fusion filter to compose the local results in creation of a global estimation.

This paper addresses the distributed estimation problem in nonlinear systems using decentralized filtering. Previous efforts on distributing filtering structures have concentrated on either decentralized filters on centralized or hierarchical topologies or essentially centralized filters on decentralized topologies based on the work of (Hassan *et al.* 1978; Speyer 1979; Willsky *et al.* 1982).

These implementations assume a truly decentralized architecture requiring no central facilities. Several decentralized results have been developed to decentralize the filter algorithm, topology, and services through tradeoffs of computation and memory that minimize communication. The result is an optimal, linear filter which needs only to share nodal dimension information between autonomous nodes. Since the Kalman filter has been an excellent means for ex-

*This work has been supported by the Hungarian Scientific Research Fund (OTKA) under grant number K 061081, which is gratefully acknowledged by the authors.

†Corresponding author. E-mail: edelmayer@sztaki.hu

‡The author is on temporary leave from University of Los Andes (ULA), Faculty of Engineering, Department of Chemical Engineering, Laboratory of Unit Operations, Mérida, Venezuela.

ploring decentralized system tradeoffs because of its optimality and linearity, the majority of solutions rely on derivations of the linear Kalman filter, see e.g., (Sun & Deng 2004).

The need for data transmission between local processors, however, is an intrinsic property of such applications. In real distributed applications, the amount of necessitated data transmission may be very large. One particular distributed filtering structure that addresses this problem is referred to *federated* filtering that is based on the information sharing principle developed in (Carlson 1988b; 1988a; 1990). Federated filtering is also known to have particularly good fault tolerant capability. These structures consists of several local filters (LF's) and a master (or fusion) filter (MF). There is a particular LF assigned to each particular subsystem. LF's work in parallel, and their solutions are periodically fused by the MF.

Although, decentralized federated structures have been extensively studied for reducing the typically high computational load of standard (centralized) filtering, its potential to fault tolerance and performance increase has not been investigated and fully realized. Moreover, whilst the techniques and theory of federated filtering (especially those based on the Kalman filter) are relatively well developed for linear systems, experiences subjecting nonlinear applications are not yet widely available. A rare exception can be found in (Ali & Fang 2005).

In addressing the problems of both fault tolerance and estimation accuracy, in this paper the applicability of an extended federated filtering idea which uses the Extended Kalman Filter (EKF) as state estimator, and a sensor redundancy management logic to maintain estimation functionality in nonlinear systems is investigated. EKF has become a standard technique of estimation in systems where the state and observation models are not linear functions of the state but instead they are general (differentiable) functions by now. EKF essentially linearizes the non-linear function around the current estimate by using standard numerical techniques. The approximation issues are sometimes treated with using polynomial approximation, see (Nørgaard 2000). In this paper the state estimation problem of a special class of systems is considered, namely, distributed systems which can be decomposed of the set of subsystems having nonlinear coupled dynamics. By the term coupled dynamics the coupling or mutual dependence of the elements of the global system state is meant.

The principle of coupling has several implications on the solution. The federated state estimators derived for the distributed filtering problem will differ providing this assumption is considered valid or not. The most important consequence of coupling is that each LF should provide an estimate of the *global* state vector based on a set of subsystem specific local measurements for best accuracy. Therefore, the local filters are not smaller dimensions than that of the centralized one in this case. However, performance gains and increased robustness makes the application of this special filter architecture favorable in the practice. Further consequences of the principle of coupled dynamics on the filtering solution will be commented later.

Another consequence of the coupling is that estimation of a particular state of the global state vector can be achieved based on several different redundant measurements. Estimations based on rearranged sensor layout in a centralized architecture may provide considerably different estimation performance. Sensor assignment, therefore, is one of the main concerns of the design in distributed systems having coupled dynamics. As a related problem of sensor redundancy, reliability of individual sensors is frequently inadequate to satisfy the stringent reliability requirements in complex safety critical systems. Therefore, an array of redundant sensors is usually employed to achieve the required system reliability through the use of redundancy. Traditionally, this idea relies on the utilization of a sensor failure detection and isolation system to detect sensor malfunctions. It is shown in this paper, how federated solutions exploit the principle of redundancy to improve both estimation accuracy and fault tolerance in the same time.

The paper is organized as follows: in Section 2, the basic features of the federating solution is briefly given. The necessary theory of the design of the federated filter is focused on the presentation of the fusion algorithm, the discussion of the EKF adopted for solving the nonlinear estimation problem is not detailed. We take the classical federated filter idea one step further by adding fault detection and sensor management to the architecture. A simulation example, demonstrating and comparing the features and performance characteristics of various solution alternatives is presented in Section 3.

Federated Filter Architecture

Federated filtering is usually regarded as a two-stage data processing technique. In the structure of filters composed of the set of local and the master filters, the LF's work in parallel, independently of one another, and their solutions are periodically fused by the MF yielding a global solution. For the solution of the nonlinear state estimation problem, in this paper the nonlinear extension of the well-known linear Kalman filtering algorithm is used in a particular structure, where both local and the master filters adopt the EKF idea for filter implementation. As both Kalman filter and the EKF considered to be standard techniques of control theory by now, only the details necessary to the discussion are mentioned here. For more information, the interested reader is referred to the literature, such as *e.g.,* (Brown & Hwang 1992) and (Mangoubi 1998).

In this work we are concerned with nonlinear dynamical systems in the nominal representation described by ordinary differential equations subject to noise

$$\begin{aligned} \dot{x}(t) &= \phi(x(t), u(t), w(t)), \\ y(t) &= h(x(t), v(t)) \end{aligned} \tag{1}$$

which can be written in state space form, by means of a set of equations of the following type

$$\begin{aligned} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x) u_i + w, \\ y_i &= h_i(x) + v, \qquad 1 \le i \le p, \end{aligned} \tag{2}$$

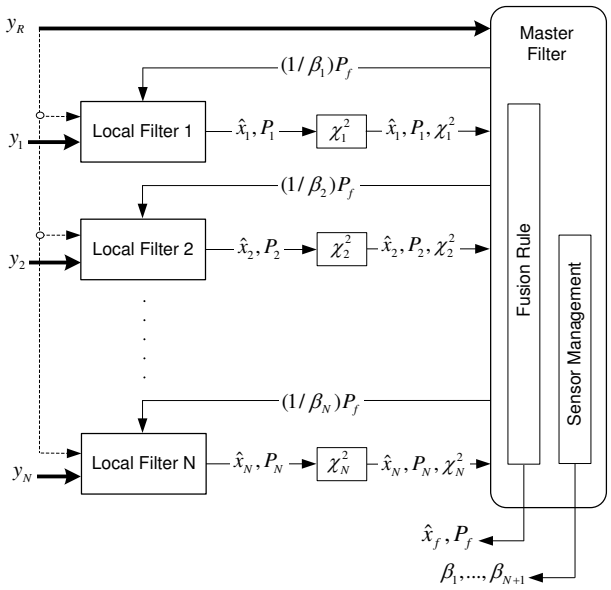


Figure 1: *The federated filter architecture. Innovation series of the local filters are subject to statistical testing (χ^2 test) for detection of changes in the noise characteristics of the measurements, upon which, the determination of the measures of the covariance information sharing β_i is relied.*

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^p$ denote respectively the state, the input and the output of the system, moreover, $w(t)$ and $v(t)$ are the system and the measurement noise. Assume each sensor s_i is measuring a signal $y_i(t)$ that is corrupted by measurement noise v_i that is a zero-mean white noise. Let Q_i and R_i denote the covariance matrix of v_i and w_i for all i, respectively

$$Q = \mathbf{E}\{v_k v_k^T\}, \quad R = \mathbf{E}\{w_k w_k^T\}. \tag{3}$$

Let the local estimate and its covariance provided by the i^{th} local filter be represented by $\hat{x}_i$ and P_i $(i = 1, 2, ..., N)$, as is shown in Fig. 1. The filtering algorithm is considered the extension of the standard linear one. Since the system is not linear the Riccati matrices that attempt to approximate the *a priori* and *a posteriori* covariances for each filter are defined, respectively, as

$$P_{k|k-1} \approx \mathbf{E}\{e_{k|k-1}\, e_{k|k-1}^T\}, \tag{4}$$

$$P_{k|k} \approx \mathbf{E}\{e_{k|k}\, e_{k|k}^T\}. \tag{5}$$

The filters are initialized with $\hat{x}_{o|o} = x_o$ and $P_{o|o} = P_o$, and then they operate recursively performing a single cycle each time a new set of measurements becomes available. Each iteration propagates the estimate from the time the last measurement was obtained to the current time. The propagation process consist of two stages: update and prediction. The update equations are responsible for the feedback, *i.e.*, for incorporating a new measurement set into the *a priori* estimate to obtain an improved *a posteriori* estimate. The *a posteriori* state estimate $x_{k|k}$ is computed as a linear combination of an *a priori* estimate $x_{k|k-1}$ and a weighted difference between an actual measurement y_k and a measurement prediction:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \left[y_k - \bar{h}(\hat{x}_{k|k-1}) \right], \tag{6}$$

$$K_k = P_{k|k-1} \bar{H}_k^T \left(\bar{H}_k P_{k|k-1} \bar{H}_k^T + R_k \right)^{-1}, \tag{7}$$

$$\bar{H}_k = \left(\frac{\partial \bar{h}(x)}{\partial x} \right)_{x=\hat{x}_{k|k-1}}$$

The matrix K_k is chosen such that the filter minimizes the *a posteriori* error covariance. The covariance matrix is updated by

$$P_{k|k} = (I - K_k \bar{H}_k) P_{k|k-1}. \tag{8}$$

The predictions equations are responsible for projecting forward, in time, the current state vector and error covariance estimates to obtain a priori estimates for the next time step. The state and covariance matrix in the next sampling instant are estimated by

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, u_k), \tag{9}$$

$$P_{k|k+1} = \bar{F}_k P_{k|k} \bar{F}_k^T + Q_k \tag{10}$$

$$\bar{F}_k = \left(\frac{\partial \bar{f}(x,u)}{\partial x} \right)_{x=\hat{x}_{k|k}, u=u_k}$$

The following characteristics distinguish the LF's and the MF. Each LF is dedicated to the measurements of local sensors. For the update process, the LF's use the update information from its local sensors only and not any others. Conversely, the MF uses the local filtered estimates $\hat{x}_i, P_i$ as quasi-observables to update the global state vector in a fusion process, sequentially. The fused estimation of the MF is usually based on measurements provided by reference sensors. Reference sensors act as fundamental sensors in the system, which are usually more dependable and have higher data rate; thus their data is often used for the initialization of some of the local filters and accountable to compensate erroneous sensor measurements relying on the decentralized architecture as will be shown later.

The MF is processed at the rate equal to the rate of the LF's which means that local outputs are subject to fusion in the next stage they are processed. The time updating solution of the MF is represented by the state estimation $\hat{x}_{N+1}$ and covariance $\hat{P}_{N+1}$, respectively. If all local estimates are uncorrelated, then the global estimate can be given as

$$P_f^{-1} = P_1^{-1} + P_2^{-1} + ... + P_N^{-1} + P_{N+1}^{-1}, \tag{11}$$

$$\hat{x}_f = P_f \left[P_1^{-1} \hat{x}_1 + ... + P_N^{-1} \hat{x}_N + P_{N+1}^{-1} \hat{x}_{N+1} \right],$$

where the inverse of the covariance matrix is called the information matrix. Eq. (11) shows that the global information is just the sum of that of the local systems. The global estimate $\hat{x}_f$ is a linear weighted combination of the local estimates with weighting matrices P_f^{-1}, P_i^{-1} $(i = 1, \ldots, N, N+1)$. However, in fact, the estimates of different local filters are correlated because of our assumption on the coupled dynamics. In order to eliminate this correlation, the process noise

and state error covariance are set to their upper bounds as follows

$$Q_i = \beta_i^{-1} Q, \qquad P_i = \beta_i^{-1} P_f, \tag{12}$$

by modifying the fusion rule (11), with $\beta_i \geq 0$ satisfying $\beta_1 + \beta_2 + ... + \beta_N + \beta_{N+1} = 1$. The parameter β is called the information-sharing factor. For the condition that makes eq. (11) holds, see (Carlson 1990).

This structure of filters can be operated in basically two different operating modes depending on how the fused process data is exploited by the local filters. When the *reset* mode is used, the master and local filters are reset by the global solution $\hat{x}_i = \hat{x}_f$, and $P_i = \beta_i^{-1} P_f$, for $i = 1, \ldots, N, N+1$. That is to say there is a continuous information feedback from master filter to local filters. In *no-reset* mode, however, this information feedback does not exist: each LF keeps its process information $(\hat{x}_i, P_i)$ produced locally, thus the MF retains none of the fused data and the global fused estimation $(\hat{x}_f, P_f)$ has no effect on any of the local estimations. There are obvious advantages and disadvantages associated with these particular resetting modes: the federated filter operated in reset mode is expected to provide better estimation accuracy, while in the no-reset mode a better tolerance of sensor faults. Note that in the special federated structure, in which each individual LF estimates the full state vector, the MF reduces to a fusion filter based on the fusion algorithm (11).

Application

Objectives

Sensor failures causing measurement drop-outs, sensor noise and measurement uncertainty will impose a practical limitation on the estimation accuracy obviously. However, the application of the federated filtering idea provides a variety of possibilities for improving filter performance. In the remainder of this paper the effect of the selection of the operating mode is investigated, based on the application example presented in the next section. The classical federated architecture is extended with sensor fault detection and a sensor management logic, in an attempt to make a tradeoff between best estimation accuracy and sensor fault tolerance in every possible time. The key to this solution is to determine how total information generated by the local filters is to be divided among the individual filters by proper assignment of the information sharing factors β_i and reset mode in the fusion filter to achieve the best possible estimation accuracy, or if a failure occurs, for higher fault tolerance.

Sensor failures may be caused by sensor drift, step changes, scale factor errors changing the mean, incorrect calibration of the sensors, etc. Correspondingly, the degraded modes of sensors can be characterized by the presence of a systematic nonzero mean, in the form of a constant jump bias, a ramp bias, or by an increase in variance of the driving noise. The diagnosis approaches can be based on the analysis of the innovation sequences of the filters. To detect failures changing the mean of the innovation sequence a number of statistical approaches are available that make use a threshold test on the moving window average of the output of the instrument. These test methods generally known as generalized likelihood ratio tests (GLRT), χ^2-tests and others, are appropriate for online implementation and embedding in the federated scheme according to Fig. 1. To simplify the presentation, this paper does not discusses these detection methods, the reader is directed to Refs. (Chien & Adams 1976; Mangoubi 1998) and also the references cited thereof.

System Model

In order to further develop the idea and demonstrate the applicability of the theory presented in the previous sections a simulation example is shown. In this example a multicomponent distillation process is considered which fractionates ternary methanol, ethanol and propanol mixture. Chemical distillation systems are among the most complex nonlinear processes. Both control and estimation of such equipments represent an engineering challenge which requires the application of sophisticated methods of advanced nonlinear control theory. Some of these systems, for instance, are controlled by Model Predictive Control (MPC) that necessitates the availability of accurate state estimation.

The process presented here consists of 30 separate distillation stages, *i.e.,* column trays, identified by the parameter N_T, plus the reboiler and a total condenser as usual. The control-input is the reflux of liquid flow rate which acts on the plant. The sensor outputs used for control and estimation purposes are the temperatures of the column tray. It is assumed that there is one temperature measurement available for each tray. The trays (and measurements) are numbered from the top to the bottom of the column. Due to the coupled nature of the nonlinear dynamics, these sensors provide a highly redundant set of measurements upon which the estimation and control of the process can be based.

The instrumentation includes temperature sensors of different quality. There are two highly reliable reference sensors used, one at the topmost and one at the bottommost of the column.

The nonlinear model of the distillation column can be represented by the following set of equipment models. The main assumptions and conditions adopted for the model development are not mentioned here, the interested reader is referred to the literature, such as e.g., (Baratti *et al.* 1998). The condenser is subject to

$$M_1 \frac{d\,(x_{i,1})}{dt} = V_s y_{i,2} - L_s x_{i,1} \left(1 + {}^1\!/_R\right) \tag{13}$$

with reflux ratio R which is normally used as control input of the distillation process. A generic tray in the enriching section (rectifying) is modeled as

$$M_j \frac{d\,(x_{i,j})}{dt} = V_s y_{i,j+1} + L_s x_{i,j-1} - V_s y_{i,j} - L_s x_{i,j} \tag{14}$$

for $i = 1, ..., N_c - 1$ and $j = 1, ..., f - 1$ where f is the tray where the column is fed. Then the feed tray is subject to

$$M_j \frac{d\,(x_{i,j})}{dt} = V_s y_{i,j+1} + L_s x_{i,j-1} - V_s y_{i,j} - L_s x_{i,j} + F_j z_{i,j}, \tag{15}$$

with feed flow rate F and molar composition z_i of component i. A generic tray in the stripping section is represented by

$$M_j \frac{d\,(x_{i,j})}{dt} = V_r y_{i,j+1} + L_r x_{i,j-1} - V_r y_{i,j} - L_r x_{i,j}, \tag{16}$$

for $j = f+1, .., N_T - 1$, and the reboiler is modeled as

$$M_{N_T} \frac{d\,(x_{i,N_T})}{dt} = L_r x_{i,N_T-1} - V_r y_{i,N_T} - L x_{i,N_T}. \tag{17}$$

In model equations (13)-(17) the parameter N_c stands for the number of liquid components, M_j is the molar holdup corresponding to a particular tray, and $y_{i,j}$ is the vapor composition of component i at stage j, which, in equilibrium with the liquid phase is given by

$$y_{i,j} = \frac{x_{i,j}\alpha_i}{\sum_{i=1}^{nc} x_{i,j}\alpha_i},$$

where α_i is the relative volatility parameter, L_s and L_r, moreover, V_s and V_r are the liquid and vapor flow rates of the stripping and rectifying sections, respectively.

The liquid compositions are considered as state variables. For a system of N_c components ($N_c = 3$ in this case), there are $N_c - 1$ state variables at each particular tray. Thus, there are 64 state variables included in the global model. The composition of the N_c^{th} component is obtained by

$$x_{N_c,j} = 1 - \sum_{i=1}^{N_c-1} x_{i,j}, \qquad 1 \le j \le N_T.$$

The temperature on a given tray is obtained using the Vapor Liquid Equilibrium equations, for simplicity this is not detailed here, see e.g., (Oisiovici & Cruz 2000). The realization of the EKF filters (both locals and master) are based on model equations (13-17). Modeling uncertainties are expected to compensate by the individual EKF's.

Simulation results

In this section, we present the results obtained for the solution of the nonlinear state estimation problem using various filter layouts and process conditions. The process simulations were performed in Matlab using the KALMTOOL package for the EKF implementations, see Ref. (Nørgaard 2000). A federated filtering scheme, resembling to the structure depicted by Fig. 1. containing three local and one master filter was derived for evaluation.

For state estimation, observability of the nonlinear system representation is required. As it was shown in Ref. (Yu & Luyben 1987), the ternary mixture distillation process is observable if at least two temperature measurements of the column are available. The performance of the estimator may improve if more than the minimum necessary measurements are used. We present the case when the minimum number of measurements is used, *i.e.,* only two of the 32 temperature measurements are available in the column for the synthesis of the filters.

Table 1: Comparison of filtering performance

Filter Layout	Sensor Fault	β_i	$\|\tilde{x}\|_2$	δ
Centralized	no fault	–	0.0134	0
Centralized	$\sigma = 10, \mu = 0$	–	0.0192	43.2
Fed. with reset	no fault	equal	0.0135	0
Fed. with reset	$\sigma = 10, \mu = 0$	equal	0.0192	43.2
Fed. w/o reset	no fault	equal	0.0136	1.5
Fed. w/o reset	$\sigma = 10, \mu = 0$	equal	0.0291	117.2
Fed. w/o reset	$\sigma = 10, \mu = 0$	$\beta_2 = 0$	0.0160	19.4
Fed. w/o reset	$\sigma = 10, \mu = 1$	equal	0.0223	66.4
Fed. w/o reset	$\sigma = 10, \mu = 1$	$\beta_2 = 0$	0.0145	8.2

Another relevant aspect of the filter design is the proper assignment of sensor locations. Not going into technical details, in our case two reference temperature measurements were allocated in the structure: one to the bottom of the column, which has the largest inertia of the system, and the other to the top of the column, so as to reflect promptly changes in the end-product composition. Relying on the above considerations the local filter estimates are based on the tray temperature measurements in the grouping LF1:(y_1, y_{31}), LF1:(y_2, y_{32}) and LF3:(y_3, y_{31}), whilst the master uses the reference inputs MF:(y_1, y_{32}). In normal operating conditions the normal and reference measurements are assumed to have sensor noise with variance $\sigma = 0.01$ and $\sigma = 0.001$, respectively, indicating the higher dependability of the reference sensors.

In simulation of the complex column behaviors, a rigorous distillation model, different from those of (13-17) based on Differential Algebraic Equations (DAE) was employed. For more details regarding this process modeling and simulation technique the reader is directed to the literature, see Ref. (Luyben 1996).

The process simulation was based on the assumption that a step change in the control signal (reflux) in zero simulation time is occurred. As an effect, the process imposes a transient behavior where the resulted state variation is the subject of the state estimation. Since the composition at the top of the column is in direct relationship with the quality of the end product (*i.e.,* methanol) of the distillation, the estimation of this composition is of the upmost importance and, therefore, the filtering results are related to the estimation of this state variable.

Two different types of sensor failures were considered. A sensor failure (shift) changing the mean value of the innovation sequence of the second measurement channel by a constant bias ($\mu = 1$), moreover increased variance in the driving noise $\sigma = 10$ instead of the normal $\sigma = 0.01$ is considered. For more accurate comparison of the results, the same noise sequences were used for each simulation experiments. Experiences for the comparison of the centralized filter performance and the federated idea were carried out. If the system operates normally, the normalized innovation sequence in a filter is a white noise sequence with a zero mean and with unit covariance matrix.

A real-time detection of sensor failures affecting the mean and/or the variance of the innovation process triggers a filter reconfiguration action in the sensor management logic of the fusion filter in an attempt to keep the performance of the impaired structure as close to the optimum as possible. This includes the modification of both the sharing factors and the reset mode of the filter.

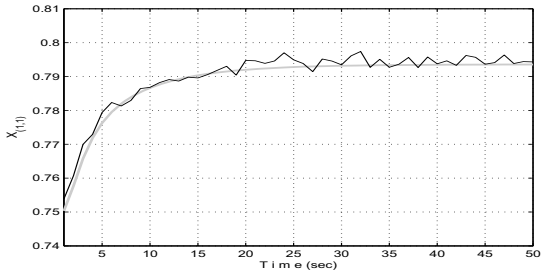


Figure 2: *Centralized estimation with no sensor fault.*

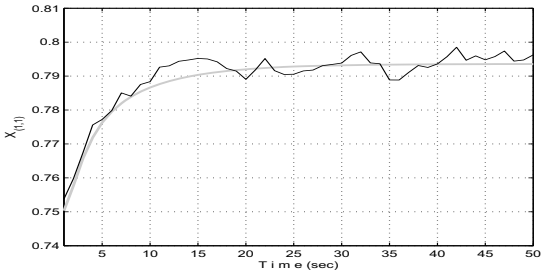


Figure 3: *Centralized estimation with increased variance in the driving noise.*

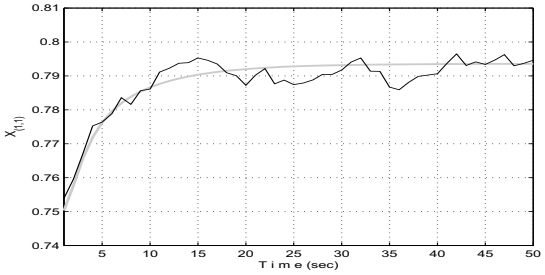


Figure 4: *Federated estimation with no reset of process information assuming increased variance and nonzero mean sensor fault in the measurement y_2 with identical sharing factors $\beta_1 = \beta_2 = \beta_3 = \beta_M$.*

A comparative summary of the results can be found in Table I showing filter performance data corresponding to particular filter configurations and various process conditions.

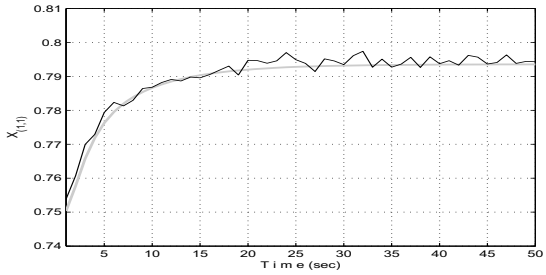


Figure 5: *Federated estimation with full reset of process information in no fault condition and identical sharing factors $\beta_1 = \beta_2 = \beta_3 = \beta_M$.*

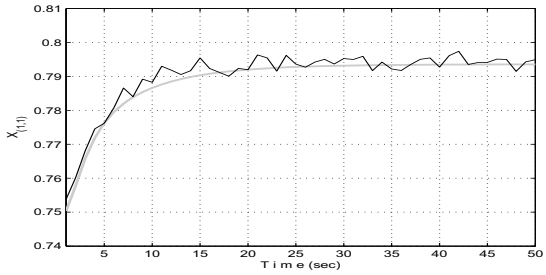


Figure 6: *Federated estimation with no reset of process information assuming increased variance in the driving noise of the measurement y_2 with sharing factor $\beta_2 = 0$.*

The performance of individual solution alternatives is characterized in terms of the L_2 norm of the estimation error $\tilde{x}$, moreover, with the relative estimation error δ given with respect to the estimation accuracy obtained with the centralized filter in fault free case.

Simulation results (see Table I) confirm that the centralized and federated filtering operated in full reset mode with identical information sharing factors provide the same performance both in faulty situation and in fault free case. Note that this result was expected, it comes from the federating theory. Differences can be experienced in faulty situations when, after detecting a fault, one switches the federating scheme to no reset mode by masking out the subjected local filter with setting the corresponding β to zero. Fig. 2-7 show the plots of some of the distinguished experiences.

Conclusion

The basic design technique of federated filtering consists of using dedicated local filters to particular components of large distributed dynamical systems and utilizing a master filter for obtaining the global estimation. The numerical simulation example presented have shown that the proper selection of the sharing factors, as well as the choice of the resetting policy of the filter may positively affect filtering performance. This support the hypothesis that a better estimation accuracy and sensor fault tolerance in the federated

solution in case of sensor degradation can be obtained.

The key to this solution is the re-organizable architecture represented by the federated idea, as well as a reconfiguration scheme that, with the aid of the utilization of the sensor fault detection and fault management logic, alters the fusion filter law to achieve a better performance. Each filter within a federated group is assigned, under the control of the fault management function of the master filter, to operate in either a full-reset or a no-reset mode. Failure of any of the input channels can be tolerated without significant deterioration of estimation performance. In case the estimation is used in a subsequent control action, operational capability of the system can be maintained. This property of the filter confirms the applicability of the idea in dependable distributed control systems.

Acknowledgment

The authors express their personal appreciation of the assistance given them in their work by Rami S. Mangoubi at Draper Lab. who provided invaluable discussions. The KALMTOOL Matlab package, written by Magnus Nørgaard at Department of Mathematical Modelling and Department of Automation, DTU Denmark, was copyrighted freeware for our research. Without their co-operation the work involved in the extensive simulation work compiling various sources of information would not have been possible.

References

Ali, J., and Fang, J. 2005. Multisensor data synthesis using federated of unscented kalman filtering. In *IEEE International Conference on Industrial Technology*, 524–529.

Baratti, R.; Bertucco, A.; Rold, A. D.; and Morbidelli, M. 1998. A composition estimator for multicomponent distillation columns -development and experimental test on ternary mixtures. *Chemical Engineering Science* 53:3601–3612.

Brown, R. G., and Hwang, P. Y. C. 1992. *Introduction to random signals and applied Kalman filtering*. Wiley, New York. Second Edition.

Carlson, N. A. 1988a. Federated filter for fault-tolerant integrated navigation systems. In *Proc. IEEE Pos. Loc. Nav. Symp.*, 110–119. Orlando, FL.

Carlson, N. A. 1988b. Information-sharing approach to federated Kalman filtering. In *Proc. IEEE National Aerospace and Electronics Conf.*, 1581. Naecon, Dayton, OH.

Carlson, N. A. 1990. Federated square root filter for decentralized parallel processes. *IEEE Trans. Aerospace and Electronic System* 26(3):517–525.

Chien, T.-T., and Adams, M. B. 1976. A sequential sensor failure detection technique and its application. *IEEE Trans. Aut. Cont.* 750–757.

Hassan, M. F.; Salut, G.; Singh, M. G.; and Title, A. 1978. A decentralized computational algorithm for the global kalman filter. *IEEE Trans. Aut. Cont.* 23(2):262268.

Luyben, L. W. 1996. *Process Modeling, simulation and control for chemical engineers*. McGraw-Hill, New York. Second Edition.

Mangoubi, R. S. 1998. *Robust Estimation and Failure Detection – A Concise Treatment*. Springer-Verlag, London.

Nørgaard, M. 2000. KALMTOOL: state estimation for nonlinear systems. Department of Mathematical Modelling and Department of Automation, DTU Denmark. 50. Tech. Report: IMM-REP-2000-6.

Oisiovici, R. M., and Cruz, S. L. 2000. State estimation of batch distillation columns using and extended kalman filter. *Chemical Engineering Science* 55:4667–4680.

Speyer, J. L. 1979. Computation and transmission requirements for a decentralized linear-quadratic-gaussian control problem. *IEEE Trans. Aut. Cont.* 24(2):266–269.

Sun, S.-L., and Deng, Z.-L. 2004. Multi-sensor optimal information fusion kalman filter. *Automatica* 40(6):1017–1023.

Willsky, A. S.; Bello, M. G.; Castanon, D. A.; Levy, B. C.; and Verghese, G. C. 1982. Combining and updating of local estimates and regional maps along sets of one-dimensional tracks. *IEEE Trans. Aut. Cont.* 27(4):799–813.

Yu, C., and Luyben, W. 1987. Control of multicomponent distillation columns using rigorous composition estimators. *I. Chem. E. Symp. Series* 104:A29–A69.

Approximate Model-Based Diagnosis Using Greedy Stochastic Search

Alexander Feldman[1] and **Gregory Provan**[2] and **Arjan van Gemund**[1]
[1]Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {a.b.feldman,a.j.c.vangemund}@tudelft.nl
[2]University College Cork, Department of Computer Science, College Road, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: g.provan@cs.ucc.ie

Abstract

Most algorithms for computing diagnoses within a model-based diagnosis framework are deterministic. Such algorithms guarantee soundness and completeness, but are NP-hard. To overcome this complexity problem, we propose a novel *approximation approach* for multiple-fault diagnosis, based on a greedy stochastic algorithm called SAFARI (StochAstic Fault diagnosis AlgoRIthm). SAFARI sacrifices guarantees of optimality, but for models in which component failure modes are defined solely in terms of a deviation from nominal behavior (known as weak fault models), it can compute 80-90% of all cardinality-minimal diagnoses, several orders of magnitude faster than state-of-the-art deterministic algorithms. We have applied this algorithm to the 74XXX and ISCAS-85 suites of benchmark combinatorial circuits, demonstrating order-of-magnitude speedup over a well-known deterministic algorithm, CDA*, for multiple-fault diagnoses.

Introduction

Model-Based Diagnosis (MBD) is an area of abductive or model-based inference in which a system model is used, together with observations about system behavior, to isolate sets of faulty components (diagnoses) that explain the observed behavior. The standard MBD formalization (Reiter 1987) frames a diagnostic problem in terms of a set of logical clauses that include mode-variables describing the nominal and fault status of system components; from this the diagnostic status of the system can be computed given an observation (OBS) of the system's sensors. MBD provides a sound and complete approach to enumerating multiple-fault diagnoses, and exact algorithms can guarantee finding an optimal diagnosis[1].

However, the biggest challenge (and impediment to industrial deployment) is the computational complexity of the MBD problem. The MBD problem of isolating multiple-fault diagnoses is known to be $\Sigma_1^{\mathbf{P}}$-complete (Bylander *et al.* 1991; Friedrich, Gottlob, & Nejdl 1990); further, the task of finding a kernel diagnosis of minimal cardinality is $\Pi_2^{\mathbf{P}}$-complete (Eiter & Gottlob 1995). Since almost all proposed MBD algorithms have been complete and exact (with some authors proposing possible trade-offs between completeness and faster consistency checking by employing methods such as BCP (Williams & Ragno 2004)), the complexity problem remains a major challenge to MBD.

To overcome this complexity problem, we propose a novel *approximation approach* for multiple-fault diagnosis, based on stochastic algorithms. SAFARI (StochAstic Fault diagnosis AlgoRIthm) sacrifices guarantees of optimality, but for diagnostic systems in which faults are described in terms of an arbitrary deviation from nominal behavior SAFARI can compute diagnoses several orders of magnitude faster than competing algorithms.

Our contributions are as follows. (1) This paper introduces an approximation algorithm for computing diagnoses within an MBD framework, based on a greedy stochastic algorithm. (2) We show the theoretical justification for the success of this algorithm, i.e., that minimal-cardinality diagnosis over weak fault models can be solved in poly-time (calling the incomplete SAT-solver BCP), and that more general frameworks are also amenable to this class of algorithm. (3) We apply this algorithm to a suite of benchmark combinatorial circuits, demonstrating order-of-magnitude speedup over a well-known deterministic algorithm, CDA*, for multiple-fault diagnoses. Moreover, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality. SAFARI is of great practical significance, as it can compute a significant fraction of cardinality-minimal diagnoses for discrete systems too large or complex to be diagnosed by existing deterministic algorithms.

The rest of this paper is organized as follows. The next section formalizes some theoretical notions in MBD. The third section presents a greedy stochastic algorithm for MBD. The fourth section shows theoretical results in support of the greedy stochastic search. It is followed by a section that discusses experimental results. Finally come conclusions and future work.

Technical Background

The discussion starts by formalizing some basic notions in MBD, extending the notions proposed by de Kleer et al. (de Kleer, Mackworth, & Reiter 1992). A *model* of an artifact is represented as a propositional $\mathbf{Wff}$ over some set

[1]Optimality has been defined in many ways in the literature, such as in terms of minimal-cardinality or most-likely diagnoses (de Kleer & Kurien 2003).

of variables V. Discerning a subset of them as failure-mode variables (*assumables*) or observable variables (*observables*) gives us a diagnostic system.

Definition 1 (Diagnostic System)**.** A diagnostic system DS is defined as the triple $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, where SD is a propositional theory describing the behavior of the system, COMPS is a set of assumable variables in SD, and OBS is a set of some observable variables in SD.

Although it is not strictly necessary, throughout this paper we will assume that $\mathrm{OBS} \cap \mathrm{COMPS} = \emptyset$.

A Running Example

The Boolean circuit shown in Figure 1 is used to give a basic idea about the intuition behind the algorithm discussed in this paper. The subtractor consists of seven components: an inverter, two or-gates, two xor-gates, and two and-gates.

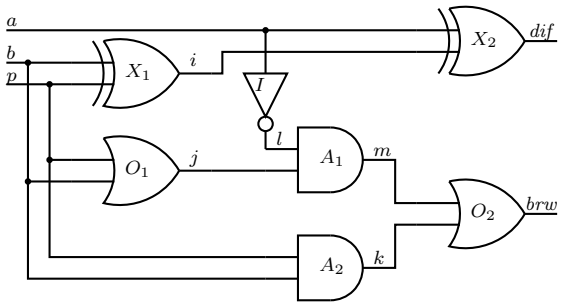


Figure 1: A subtractor circuit.

The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models an inverter, where the variables i, o, and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2)$ and an or-gate is $h \Rightarrow (o \Leftrightarrow i_1 \vee i_2)$. Finally, an xor-gate is specified as $h \Rightarrow (o \Leftrightarrow \neg(i_1 \Leftrightarrow i_2))$.

These propositional formulae are copied for each gate in Figure 1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus receiving a propositional formula for SD:

$$\mathrm{SD} = \begin{cases} X_1 \Rightarrow (i \Leftrightarrow \neg(b \Leftrightarrow p)) \\ X_2 \Rightarrow (dif \Leftrightarrow \neg(a \Leftrightarrow i)) \\ O_1 \Rightarrow (j \Leftrightarrow b \vee p) \\ O_2 \Rightarrow (brw \Leftrightarrow m \vee k) \\ A_1 \Rightarrow (m \Leftrightarrow l \wedge j) \\ A_2 \Rightarrow (k \Leftrightarrow b \wedge p) \\ I \Rightarrow (a \Leftrightarrow \neg l) \end{cases}$$

The set of component (assumable) variables is $\mathrm{COMPS} = \{X_1, X_2, O_1, O_2, A_1, A_2, I\}$. The set of observable variables is $\mathrm{OBS} = \{a, b, p, dif, brw\}$.

Minimal Diagnosis

The traditional query in MBD results in finding terms of assumable variables which are explanations for the system description and an observation. The first definition of diagnosis uses a set notation.

Definition 2 (Diagnosis)**.** A *diagnosis* for the system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, given an observation term α over the variables in OBS, is a set $D \subseteq \mathrm{COMPS}$ such that:

$$\mathrm{SD} \wedge \alpha \wedge \left[\bigwedge_{c \in D} \neg h_c \right] \wedge \left[\bigwedge_{c \in (\mathrm{COMPS} \setminus D)} h_c \right] \not\models \bot$$

In the MBD literature, a range of types of "preferred" diagnosis have been proposed. In the following, we assume a preference relation $\preceq$ that assigns a partial order over the set of diagnoses.

The first ordering we consider is a subset-ordering $\preceq_S$:

Definition 3 (Subset-Minimal Diagnosis)**.** A diagnosis D is subset-minimal, i.e., $D \preceq_S D'$, if no proper subset $D' \subset D$ exists such that D' is also a diagnosis.

Throughout this paper we interchangeably use a propositional notation for expressing diagnoses. In this case we simply construct a conjunction of literals, each literal having a negative sign if its respective variable is in D and a positive sign otherwise. Consider the example from Figure 1 and an observation $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$. In this case $D_1 = \{A_1, A_2, X_1\}$ is a diagnosis and $D_2 = \{A_1, X_1\}$ is a minimal diagnosis (there are seven more minimal diagnoses for $\mathrm{SD} \wedge \alpha$). Alternatively, instead of D_1 we may write $D'_1 = \neg X_1 \wedge X_2 \wedge O_1 \wedge O_2 \wedge \neg A_1 \wedge \neg A_2 \wedge I$.

The *cardinality* of a diagnosis D is the size of D and is denoted as $|D|$. It represents the number of faulty components in COMPS given SD and α. Next to computing minimal diagnoses, it is of interest to MBD to compute some or all minimal-cardinality diagnoses, given a diagnostic problem.

Definition 4 (Cardinality-Minimal Diagnosis)**.** A diagnosis D is a minimal-cardinality diagnosis if it is a minimal diagnosis and no other diagnosis D' exists such that $|D| < |D'|$.

The cardinality of a cardinality-minimal diagnosis computed from a system description SD and an observation α is denoted as $\mathit{MinCard}(\mathrm{SD} \wedge \alpha)$. For our example and the observation $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$, it follows that $\mathit{MinCard}(\mathrm{SD} \wedge \alpha) = 2$. Note that in this case all minimal diagnoses are also cardinality-minimal diagnoses.

Note that there are subset-minimal diagnoses which are not cardinality-minimal diagnoses. Consider, for example, the diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, where $\mathrm{SD} = (h_1 \wedge h_2 \wedge x) \vee (h_4 \wedge x)$, $\mathrm{COMPS} = \{h_1, h_2, h_3, h_4\}$, $\mathrm{OBS} = \{x\}$, and $\alpha = x$. In this case, $D_1 = \{h_1, h_2, h_3\}$ is a non-subset-minimal diagnosis, $D_2 = \{h_1, h_2\}$ and $D_3 = \{h_4\}$ are subset-minimal diagnoses, but only D_3 is a cardinality-minimal diagnosis.

Another important diagnosis preference relation that we consider is one induced by a probability distribution over the failure modes, $Pr : \mathrm{COMPS} \rightarrow [0, 1]$. If we assume that all components fail independently, then the probability of a multiple-fault F is just the product of the component probabilities, i.e., $Pr(F) = \prod_{F_i \in F} Pr(F_i)$. We assume that Pr has an associated ordering relation $\preceq_{Pr}$. We further assume that we have a non-trivial probability assignment to fault modes, i.e., that there are no assignments of probabilities of 0 or 1, in which case the fault status of that component is fixed *a priori*.

Definition 5 (Probability-Minimal Diagnosis)**.** Given a non-trivial probability assignment to component failure modes, a probability-maximal diagnosis ω is a fault-mode such that $\nexists$ any other diagnosis ω' such that $Pr(\omega') > Pr(\omega)$.

It is simple to show that a subset-ordering $\preceq_S$ holds whenever a non-trivial probability assignment exists, i.e., for diagnoses D_1 and D_2, $D_1 \preceq_S D_2 \Rightarrow D_1 \preceq_{Pr} D_2$.

It is also simple to show that the cardinality-ordering $\preceq_C$ holds iff a corresponding non-trivial probability assignment exists, i.e., if for any components $C_i,\ C_j \in COMPS$, $C_i \preceq_C C_j \Leftrightarrow C_i \preceq_{Pr} C_j$, then for diagnoses D_1 and D_2, $D_1 \preceq_S D_2 \Leftrightarrow D_1 \preceq_{Pr} D_2$.

In the following, we will focus on subset-minimal and cardinality-minimal diagnoses; these two relationships mean that our results will also hold for a probability-ordering $\preceq_{Pr}$.

Fault Models

MBD defines two broad classes of fault models, based on weak and strong modeling assumptions for abnormal behavior.

Weak-fault models define normative behavior of their components only, i.e., models which specify no fault-modes.

Definition 6 (Weak Fault Model)**.** A diagnostic system DS belongs to the class **WFM** iff SD is in the form $(h_1 \Rightarrow F_1) \wedge \ldots \wedge (h_n \Rightarrow F_n)$ such that for $1 \leq i, j \leq n$, $\{h_i\} \subseteq$ COMPS, $F_j \in$ **Wff**, and none of h_i appears in F_j.

In contrast, *strong fault models* specify the faulty behavior of their components. One way to define such a model is by partitioning the set of observable variables OBS into two subsets IN and OUT (denoting input and output variables respectively). Defining values to IN and COMPS then allows us to find a *unique assignment* to the values in OUT.

Definition 7 (Strong Fault Model)**.** Given a system DS and a partition OBS $=$ IN $\cup$ OUT, DS $\in$ **SFM** if for any instantiation ϕ of all variables in IN $\cup$ COMPS, it holds that there is exactly one term ψ such that $\phi \models$ SD $\wedge\ \psi$ and ψ is an instantiation of all variables in OUT.

In the following we show how our stochastic algorithm can compute subset- and cardinality-minimal diagnoses for SD such that SD $\in$ **WFM**, and indicate how the algorithm can be generalized to cover SD $\in$ **SFM**.

Stochastic MBD Algorithm

In this section we discuss an algorithm for computing multiple-fault diagnoses using stochastic search.

A Simple Example (Continued)

We now show what happens when we apply A* and stochastic search to our running example.

A deterministic A* search for the above diagnosis discovers it after expanding 127 nodes and performing 19 consistency checks. Even enabling conflict focusing may not result in a small number of generated nodes and consistency checks (this depends on the model, observation and conflict extraction mechanism), which shows how deterministic diagnosis search becomes impractical with bigger systems.

We will now show a two-step diagnostic process that requires fewer variable assignments and consistency checks. Step 1 involves randomly choosing candidates. Step 2 attempts to minimize the fault cardinality in these candidates.

In step 1, the stochastic diagnostic search for the subtractor example will start from a random quintuple candidate[1]. In this particular version of our algorithm, once a component is marked as healthy, it cannot be changed back to faulty. To compensate for that, we perform multiple restarts from a random candidate. In our subtractor example and for $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$, if $X_1 \wedge X_2$ is in the initial "guess" it will prove inconsistent with SD $\wedge\ \alpha$ and another quintuple fault candidate will be guessed.

Assume that the second candidate is $\omega_2' = \neg X_1 \wedge \neg X_2 \wedge O_1 \wedge \neg O_2 \wedge \neg A_1 \wedge \neg A_2 \wedge I$. Clearly, SD $\wedge\ \alpha \wedge \omega' \not\models \bot$. The search algorithm may next try to improve the diagnosis by "flipping" A_2. The candidate $\omega_3' = \neg X_1 \wedge \neg X_2 \wedge O_1 \wedge \neg O_2 \wedge \neg A_1 \wedge A_2 \wedge I$ is a valid quadruple fault diagnosis and it can be improved twice more by "flipping" X_2 and O_2. This gives us the final double-fault $\omega_6' = \neg X_1 \wedge X_2 \wedge O_1 \wedge O_2 \wedge \neg A_1 \wedge A_2 \wedge I$. The actual algorithm is somewhat more involved as during the variable flipping it is normal to find inconsistencies. Instead of restarting, it will simply discard these inconsistent candidates until some termination criterion is satisfied.

Intuitively, from our example, due to the large number of double fault diagnoses explaining the same observation, it is not difficult to randomly guess sequences of variables which need to be false in order to explain the observation.

A Greedy Stochastic Algorithm for Computing Cardinality-Minimal Diagnoses

The greedy stochastic algorithm, which we introduce next, finds multiple cardinality-minimal diagnoses (if such exist).

The stochastic algorithm presented in this paper uses the previously-described extension to DS (Provan 2005), a valuation function Pr : COMPS $\rightarrow [0, 1]$. In the analysis of our algorithm we use Pr to determine if an assignment to a health variable denotes a failure or a healthy mode. The performance of the algorithm presented in this paper is not sensitive to Pr and the only use of the probabilities is to guide the search for more efficient performance.

The randomized search process performed by SAFARI has two parameters, M and N. There are N independent searches that start from randomly generated candidates. After an initial candidate ω is found to be consistent with SD $\wedge\ \alpha$, i.e., ω is a diagnosis, the algorithm tries to improve the cardinality of the diagnosis (while preserving its consistency) by randomly "flipping" fault literals.

Each attempt to find a cardinality-minimal diagnosis terminates after M unsuccessful attempts to change the value of a fault variable to healthy state. Thus, increasing M will lead to a better exploitation of the search space and possibly diagnoses of lower cardinality, while decreasing it will improve the overall speed of the algorithm.

[1] In the formal description of the algorithm we describe a method for determining the initial candidates.

Algorithm 1 SAFARI: A stochastic hill climbing algorithm for approximating a set of cardinality-minimal diagnoses.

```
1: function HILLCLIMB(DS, α, M, N, Pr) returns a trie
       inputs: DS = ⟨SD, COMPS, OBS⟩, a diag. system
               α, term, observation
               M, integer, climb restart limit
               N, integer, number of tries
               Pr, a valuation function
       local variables: m, n, integers
                        ω, ω′, terms
                        R, a trie
2:     n ← 0
3:     while n < N do
4:         ω ← RANDOMCANDIDATE(Pr)
5:         if SD ∧ α ∧ ω ⊭⊥ then
6:             m ← 0
7:             while m < M do
8:                 ω′ ← IMPROVEDIAGNOSIS(Pr, ω)
9:                 if SD ∧ α ∧ ω′ ⊭⊥ then
10:                    ω ← ω′
11:                    m ← 0
12:                else
13:                    m ← m + 1
14:                end if
15:            end while
16:            unless ISSUBSUMED(R, ω) then
17:                ADDTOTRIE(R, ω)
18:                REMOVESUBSUMED(R, ω)
19:            end unless
20:            n ← n + 1
21:        end if
22:    end while
23:    return R
24: end function
```

Similar to deterministic methods for MBD, SAFARI uses a SAT-based procedure for checking the consistency of $\mathrm{SD} \wedge \alpha \wedge \omega$. Because SD and α do not change in consistency checks, using an LTMS (McAllester 1990) can improve search efficiency. The implementation of SAFARI combines a BCP-based LTMS to check for inconsistencies. If a candidate is consistent, a second DPLL-based check is invoked for completeness.

The RANDOMCANDIDATE function generates a candidate diagnosis, used for "seeding" the diagnostic search. The valuation function can be modified to provide a more informed starting point, thus decreasing the number of "climbing" steps. The initial diagnosis ω should be of high cardinality, to increase the likelihood of $\mathrm{SD} \wedge \alpha \wedge \omega \not\models \bot$. In order to do that, we generate an instantiation of ω by using Pr and *scaling* the a priori probabilities in Pr to bias the pdf from which we draw the initial candidates. Consider an example in which each component $h \in \mathrm{COMPS}$ fails with a probability of 5%. The valuation function is $Pr(h = \mathbf{False}) = 0.05$. We may use a scaling coefficient $k = 5$ which would lead to RANDOMCANDIDATE returning a candidate with a quarter of the components failing.

The biasing of Pr can improve the efficiency of Algorithm 1 by exploiting knowledge about the likelihood of the cardinality of the cardinality-minimal diagnosis. In particular, when expecting cardinality-minimal diagnoses of high cardinality Pr should be configured to return an initial fault of higher cardinality. If the expected faults are of small cardinality, the search may start from a candidate with fewer faulty components, in which case more attempts (increased N) would be necessary to find local diagnoses close to the global optimum.

The IMPROVEDIAGNOSIS function generates a candidate ω' of smaller cardinality than the diagnosis ω, supplied as an argument. This is done by flipping a random faulty literal in ω. The probability of flipping a faulty literal l in ω is inverse proportional to the a priori probability $Pr(l)$. Consider a diagnosis $\omega = \neg h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge \neg h_4$, where $Pr(h_1 = \mathbf{False}) = Pr(h_2 = \mathbf{False}) = 0.1$ and $Pr(h_3 = \mathbf{False}) = Pr(h_4 = \mathbf{False}) = 0.025$. In this case IMPROVEDIAGNOSIS would return $\omega' = h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge \neg h_4$ or $\omega' = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge \neg h_4$, each of the two with probability of 0.4, and $\omega' = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge \neg h_4$ or $\omega' = \neg h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge h_4$ the latter with probability 0.1.

There is no guarantee that two diagnostic searches, starting from a random diagnoses, would not lead to the same cardinality-minimal diagnosis. To prevent this, we store the generated diagnoses in a trie R, from which it is straightforwards to extract the resulting diagnoses by recursively visiting its nodes. A diagnosis ω is added to the trie R by the function ADDTOTRIE, iff no subsuming diagnosis is contained in R (the ISSUBSUMED subroutine checks on that condition). After adding a diagnosis ω to the resulting trie R, all diagnoses contained in R and subsumed by ω are removed by a call to REMOVESUBSUMED. The workings of the trie functions as well as a thorough description of the trie data structure can be found in (Forbus & de Kleer 1993).

Optimality and Complexity of Greedy Stochastic Algorithm

One of the key factors in the success of the proposed algorithm is the exploitation of the continuity of the search-space of weak fault models, where by continuity we mean that we can monotonically reduce the cardinality a non-minimal diagnosis. This section shows that our algorithm can guarantee finding minimal diagnoses in weak fault models in polynomial time (given a SAT oracle such as BCP), and trades off optimality in more general diagnostic frameworks, such as cardinality-minimal diagnostic inference or strong fault models.

Cardinality-Minimal Diagnosis in Weak-Fault Models

We will show some properties of minimal diagnoses. These properties are also true for fault-modes with a slight adaptation of the notation.

Hypothesis 1 (Minimal Diagnosis Hypothesis)**.** Let SD be a system description and D' a diagnosis. The Minimal Diagnosis Hypothesis (MDH) holds in SD if for any D such that $D \supset D'$ it holds that D is also a diagnosis.

It has been shown in (de Kleer, Mackworth, & Reiter 1992) that if a model $\mathrm{SD} \in \mathbf{WFM}$ (cf. Definition 6), then the Minimal Diagnosis Hypothesis (MDH) holds. There are other theories $\mathrm{SD}' \notin \mathbf{WFM}$ for which MDH holds. Unfortunately, no necessary condition is known for MDH to hold in an arbitrary SD'.

Using MDH together with Definition 3, if $\mathrm{SD} \in \mathbf{WFM}$ then we immediately have the following lemma.

Lemma 1. *If D is a diagnosis of a diagnosis problem* SD *and MDH holds for* SD*, then there is a subset-minimal diagnosis D' that subsumes D, i.e., $D' \subseteq D$.*

Our greedy algorithm starts with a "seed" diagnosis and then randomly flips faulty component variables. We now use the MDH property to show that, starting with a non-minimal diagnosis D, the greedy stochastic diagnosis algorithm can monotonically reduce the size of "seed" diagnosis to obtain a minimal diagnosis through appropriately flipping a fault variable from faulty to healthy; if we view this flipping as search, then this search is continuous in the diagnosis space.

Theorem 1. *Given a a weak fault model* $\mathrm{SD} \in \mathbf{WFM}$ *and a non-subset-minimal diagnosis D with $|D| = \mu \leq n$ faulty components, the greedy stochastic diagnosis algorithm is guaranteed to compute a minimal diagnosis.*

Proof. Assume that we have a model with n components, and we start our diagnostic inference by choosing a (non-subset-minimal) diagnosis D with $|D| = \mu < n$ faulty components (assumable variables). We first show that we compute a subsumed diagnosis D' after each step, where a step includes a variable flip and consistency check. If we randomly flip κ mode variables from faulty to healthy and if D' is consistent, we obtain a diagnosis D' with $|D'| = \mu - \kappa$ faulty variables. Hence, for any diagnosis D' obtained in this manner, by the non-minimality of D (using the MDH property), we know that $D' \subseteq D$.

Through a simple inductive argument, we can continue this process until we obtain a minimal diagnosis. □

We can now prove the following correctness result:

Theorem 2. *The greedy stochastic diagnosis algorithm is guaranteed to compute a subset-minimal diagnosis for a weak fault model with $|\mathrm{OBS}| = n$ in $O(\Theta n)$ time, where $O(\Theta)$ is the complexity of a consistency check.*

Proof. By Theorem 1, we are guaranteed to compute a subset-minimal diagnosis starting with a diagnosis of cardinality k. There is always a diagnosis that we can start from, as we can start with a diagnosis of cardinality $k = n$, since the assignment with all fault variables set to faulty is always consistent in a weak fault model.

It is simple to show that, starting with a seed diagnosis of cardinality k, we can compute a subset-minimal diagnosis in $O(k)$ steps, where a step includes a variable flip and consistency check. In the worst case, we can have an n-fault diagnosis. Hence, the total complexity is then $O(\Theta n)$ time, where $O(\Theta)$ is the complexity of a consistency check. □

Since consistency-checking for this model class can be done in polynomial time (using BCP propagation), we have just demonstrated a polynomial-time algorithm for computing minimal diagnoses in a weak-fault-model system description (MDWFM). This result has been shown in the literature for a divide-and-conquer approach (Mozetič 1992), but to our knowledge no current algorithm takes advantage of this approach. It is also expected that MDWFM should be easier than more general forms of diagnostic inference since the worst-case result of MDWFM is $\Pi_1^{\mathbf{P}}$-complete (Friedrich, Gottlob, & Nejdl 1990), whereas the more general task of finding a kernel diagnosis of minimal cardinality is $\Pi_2^{\mathbf{P}}$-complete (Eiter & Gottlob 1995).

More General Diagnostic Frameworks

This section now addresses how the algorithm will perform in more general diagnostic frameworks, such as computing cardinality-minimal diagnoses or dealing with strong fault models.

In generalizing beyond subset-minimal diagnosis computation, the literature indicates that most other definitions are computationally harder; indeed, as noted above, diagnostic inference for these more general cases is intractable (Friedrich, Gottlob, & Nejdl 1990; Eiter & Gottlob 1995).

We can modify SAFARI to compute a wide variety of diagnoses, e.g., subset-minimal, non-minimal, cardinality-minimal, etc., by modifying the type of diagnosis computed together with the trie maintenance and subsumption testing of lines 16-18 of the pseudo-code. It is thus trivial to tweak the behavior of SAFARI to output only the class of diagnosis that is of interest for the particular application.

The complexity of SAFARI is derived in a straightforward way.

Proposition 1. *The time complexity of Algorithm 1 is $O(MN \log N\Theta)$, where Θ is the complexity of the consistency checking procedure.*

The $\log N$ factor comes from the trie maintenance, which contains a maximum number of N diagnoses with some ordering imposed on their literals. Note that the average case complexity of consistency checking, although exponential in the worst case, is low polynomial when incomplete methods like BCP are used (Zabih & McAllester 1988) or when the model is highly-observable.

In this more general case we have no completeness guarantee, as we do for subset-minimal diagnoses. Note that this is effectively a polynomial-time algorithm that trades off some small amount of completeness and optimality for significant improvements in efficiency relative to deterministic diagnosis algorithms. In these more general frameworks, from a theoretical perspective one can only provide probabilistic arguments about the likelihood of finding particular classes of diagnosis; this is a topic of future work. The experimental results presented later in this article show that significant speedups over complete algorithms are possible while losing relatively little diagnostic completeness.

In generalizing from weak to strong fault models, the key difference is the increased difficulty in "guessing" the initial diagnosis for SAFARI. For a weak fault model, we are guaranteed to find a subset-minimal diagnosis by choosing an

initial diagnosis with all components faulty,[2] but this guess is not guaranteed to be consistent in a strong fault model. Developing a robust diagnosis initialization algorithm for strong fault models is a topic for future research.

Experimental Results

The next section discusses some empirical results measured from an implementation of the algorithm shown in this paper. In the following, all models are weak fault models, i.e., $\mathrm{SD} \in \mathbf{WFM}$.

Implementation Notes and Test Set Description

We have implemented SAFARI in approximately 700 lines of C code (excluding the LTMS and DPLL consistency checking) and it is a part of the LYDIA[3] package.

Table 1 summarizes the benchmark suite. All models are derived from the 74XXX and ISCAS85 family of benchmark circuits. We have added an assumable variable to each gate in each model. The benchmark implements weak fault models for each component, in a way similar to the example. The same valuation function Pr has been used in all the experiments. In particular, $Pr(h = \mathbf{False}) = 0.01$, and $Pr(h = \mathbf{True}) = 0.99$.

Name	Description	H	V	C_w	O
74182	4-bit CLA	19	47	75	14
74283	4-bit adder	40	89	130	14
74L85	4-bit comparator	41	93	134	14
74181	4-bit ALU	62	138	216	22
c432	27-channel int. controller	160	356	514	43
c499	32-bit SEC circuit	202	445	714	73
c880	8-bit ALU	383	826	1 112	86
c1355	32-bit SEC circuit	546	1 133	1 610	73
c1908	16-bit SEC/DEC	880	1 793	2 378	58
c2670	12-bit ALU	1 193	2 543	3 269	221
c3540	8-bit ALU	1 669	3 388	4 608	72
c5315	9-bit ALU	2 307	4 792	6 693	301
c6288	32-bit multiplier	2 416	4 864	7 216	64
c7552	32-bit adder	3 512	7 230	9 656	313

Table 1: Test model sizes.

Table 1 shows the basic characteristics of the ISCAS-85 and 74XXX models. The number of assumable variables is denoted as H. The total number of variables is denoted V and the number of clauses in the CNF representation is denoted as C_w. The number of observable variables is denoted as O.

All the experiments described in this paper are performed on a host with 1.86 GHz Pentium M CPU and 2 Gb of RAM.

Comparison to HA* and Multiple-Fault Scalability

We compare the performance of SAFARI to HA* (Feldman & van Gemund 2006) with Max-Fault Min-Cardinality (MFMC) observation vectors (Feldman, Provan, & van Gemund 2007). MFMC observation vectors maximize the number of faults in the cardinality-minimal diagnoses consistent with a model. From the deterministic algorithms, HA* performs better than CDA* (Williams & Ragno 2004) in finding cardinality-minimal diagnoses of high cardinality. On the other hand, CDA* is very fast in finding faults of small cardinality (single and double faults) and we will then compare SAFARI to CDA*.

In our first experiment we use small models with observations maximizing the number of faults in a cardinality-minimal diagnosis. The results, shown in Table 2, illustrate the most important advantage of the stochastic algorithm: its performance does not degrade when the fault cardinality increases. We have run two groups of experiments: finding a single cardinality-minimal diagnosis and finding all cardinality-minimal diagnoses.

Next we describe the notation in the column headings of Table 2. MFMC is the number of faults in the cardinality-minimal diagnosis consistent with the MFMC observation. T_h is the time for finding a single diagnosis by the HA* algorithm. T is the time for finding a single diagnosis by Algorithm 1. C is the cardinality of the diagnosis generated by SAFARI. K is the number of cardinality-minimal diagnoses as counted by the deterministic algorithm HA*. The time for finding all of these diagnoses by HA* is denoted as T'_h. T' is the time for SAFARI to find multiple cardinality-minimal diagnoses. The number of such diagnoses is denoted as K'.

Note that SAFARI is not designed to find a single cardinality-minimal diagnosis. In these experiments, we have simply configured it with a number N of runs in order to return a small number of diagnoses, and we have ignored all but the first diagnosis. We performed the single fault experiments, shown in Table 2, with $N = 8$ and $M = 4$. For the multiple diagnoses, the algorithm is configured with $M = M'$ and $N = N'$ as described in Table 2.

We have averaged the results of all the experiments involving SAFARI over 10 runs. SAFARI is a local search algorithm, and hence it can compute a suboptimal diagnosis. This was the case in the 74283 model experiments, in which 5 out of 10 runs returned a cardinality-minimal diagnosis with 6 faults, while the global optimum has 5 faults, resulting in the 5.5 value for C in Table 2.

Table 2 demonstrates the main advantage of SAFARI, that its performance does not depend on the number of faults in the cardinality-minimal diagnoses. Furthermore, SAFARI finds a good coverage of all cardinality-minimal diagnoses. This coverage varies from 81% in 74182 to 90% in 74L85.

We have also run SAFARI on bigger circuits. Figure 2 shows the time for finding multiple-fault diagnosis for c880 and c7552. The diagnosis was run k times where k is the number of outputs in each circuit. For run $x = 0$, we have assigned random values to the inputs and computed (by using propagation) the values of all the outputs. For $x = 1$ we have flipped one output in α, for $x = 2$ two outputs, etc. Thus on the horizontal axis in Figure 2 we have the Hamming distance between α and an observation consistent with a no-fault (nominal) diagnosis. Again, SAFARI showed no

[2] In SAFARI we start with half the components faulty and use the parameter M to restart if our initial guess is incorrect.

[3] LYDIA (including an implementation of SAFARI) can be downloaded from `http://fdir.org/lydia/`.

		Single Diagnosis				Multiple Diagnoses				
Name	MFMC	T_h	T	C	K	T'_h	T'	K'	N'	M'
74182	5	38	2	5	300	171	143	242	800	4
74283	5	4 708	4	5.5	814	27 606	5759	665	10 000	4
74L85	3	143	4	3	100	1 281	184	90	400	4
74181	7	106 386	9	7	3 817	634 739	31 377	3 236.7	20 000	8

Table 2: Times [ms] for diagnosing MFMC faults by SAFARI and HDA*.

dependency on α, only a small increase in the diagnostic time for c7552 due to the difficulty of finding an initial diagnosis. The latter can be easily overcome by scaling the initial a-priori probabilities.

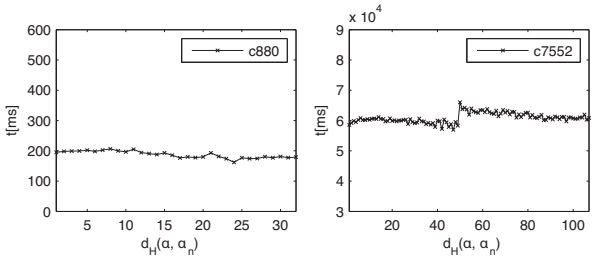


Figure 2: Diagnosis time of SAFARI with multiple observation vectors.

Comparison to CDA*

Table 3 shows the result from finding single and double faults with arbitrary (manually computed) observations. Finding single faults is known to be trivial in MBD and CDA* performs well on these simple problems. The time for finding a single fault by CDA* is shown in column T_1^*. The time for the CDA* algorithm to find a double fault is shown in column T_2^*. The CDA* algorithm could not compute a double fault diagnosis in less than 10 min time for the five biggest circuits, in which cases we have interrupted the search.

	Single-Fault			Double-Fault		
Name	T_1^*	T_1	C_1	T_2^*	T_2	C_2
c432	9	32	1	5	34	2
c499	3	53	1	152	64	2
c1908	34	95	1	509	94	2
c880	18	186	1	62 068	186	2
c1355	11	285	1	4 300	310	2
c2670	1 425	1 362	1	–	1 352	2
c3540	3 050	3 080	1	–	3 115	2
c5315	13 849	19 322	1	–	19 764	2
c6288	18 317	11 070	1.4	–	11 366	2.2
c7552	35 801	37 269	1	–	37 585	2.2

Table 3: Running times [ms] of CDA* and SAFARI.

The times for the stochastic algorithm to discover a single and a double fault are denoted as T_1 and T_2 respectively. We have used $M = 4$ and $N = 8$ for the search, that is, maximum number of four retries before giving up the climb, and a total of 4 attempts. In some of the cases, our stochastic algorithm could not find a cardinality-minimal diagnosis, but a suboptimal one. We have shown the cardinality of the results for single and double faults in columns C_1 and C_2, respectively. Again, the values of T_1, C_1, T_2, and C_2 are averaged over 10 runs.

The relatively small number of restarts lead to small overall search time and in a very few cases to suboptimal result for the diagnosis cardinality. Increasing N would lead to finding a global cardinality-minimal diagnosis in all the cases. We note that increasing M would not help to finding a diagnosis of lower cardinality.

As is visible from Table 3, in the single fault scenario, CDA* performs better than the stochastic algorithm, which is not surprising as in CDA* all single fault candidates are tested first. On the other hand, the stochastic method performed 8 independent attempts to find a cardinality-minimal diagnosis which, having the overhead of consistency checking, led to the slightly worse performance for computing single fault diagnoses.

Similar to the earlier experiments, the performance of SAFARI does not degrade when the number of faults increases. This is not the case with deterministic algorithms like CDA* or HA*. The time for the stochastic algorithm to find a double fault is the same as for finding a single fault, while CDA* suffers from a combinatorial explosion.

Our experiments show that, in most cases, the stochastic algorithm finds diagnoses of near-optimal cardinality in time orders of magnitude faster than state-of-the-art deterministic algorithms. Further, the diagnostic time of the stochastic algorithm is not affected by the number of faults in the cardinality-minimal diagnosis, which is certainly not the case with the two deterministic algorithms. The only case in which the stochastic algorithm performs slightly worse than a deterministic one, is with single fault diagnoses.

Related Work

Our proposed approach differs significantly with almost all model-based diagnosis algorithms that appear in the literature. While most advanced MBD algorithms make use of preferences, e.g., fault-mode probabilities, to improve search efficiency, the algorithms themselves are deterministic, and use the preferences to identify the most-preferred solutions. This contrasts with stochastic SAT algorithms, which rather than backtracking may randomly flip variable assignments to determine a satisfying assignment.

The most closely-related diagnostic approach is that of

Vatan et al. (Vatan *et al*. 2003), who map the diagnosis problem into the monotone SAT problem, and then propose to use efficient SAT algorithms for computing diagnoses. The approach of Vatan et al. has shown speedups in comparison with other diagnosis algorithms; the main drawback is the number of extra variables and clauses that must be added in the SAT encoding, which is even more significant for strong fault models and multi-valued variables. In contrast, our approach works directly on the given diagnosis model and requires no conversion to another representation.

The MBD problem is different than SAT in that it is an optimization problem, i.e., one typically wants to find a minimal diagnosis, using some minimality criterion such as cardinality-minimal diagnosis. Hence one cannot map the diagnosis problem directly to SAT. We can show an encoding whereby one can use cardinality constraints (Bailleux & Boufkhad 2003) to encode a notion of minimal diagnosis, and then adopt SAT algorithms.

Stochastic algorithms have been discussed in the framework of constraint satisfaction (Freuder *et al*. 1995) and Bayesian network inference (Kask & Dechter). The latter two approaches can be used for solving suitably translated MBD problems. It is often the case, though, that these encodings are more difficult for search than specialized ones.

Conclusion and Future Work

We have described a greedy stochastic algorithm for computing diagnoses within a model-based diagnosis framework. We have shown that subset-minimal diagnoses can be computed optimally in weak fault models, and that almost all cardinality-minimal diagnoses can be computed for more general fault models.

We have applied this algorithm to a suite of benchmark combinatorial circuits encoded using weak fault models, and shown significant performance improvements for multiple-fault diagnoses, compared to a well-known deterministic algorithm, CDA*. Our results indicate that, although the greedy stochastic algorithm is outperformed for the single-fault diagnoses, it shows at least an order-of-magnitude speedup over CDA* for multiple-fault diagnoses. Moreover, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality.

We have demonstrated the superior performance (over deterministic algorithms) of SAFARI for the class of discrete circuits specified using weak fault models. We argue that SAFARI can be of broad practical significance, as it can compute a significant fraction of cardinality-minimal diagnoses for systems too large or complex to be diagnosed by existing deterministic algorithms.

This paper raises a number of questions, which we plan to address in future work. On the theoretical side, we have shown that weak-fault models are solved optimally for several fault- and model-definitions. We expect semi-weak models, for which nominal behavior and *some* failure modes are specified, to be dominant in fault-modeling. We plan a more extensive theoretical investigation of such fault distributions when empirical data from these cases is collected.

On the algorithmic side, we plan to experiment with a wider variety of stochastic methods. These include simulated annealing, genetic search and others. The algorithmic work would benefit from an extensive set of benchmarking models, coming not only from digital circuits, but random models, real-world models and others. Last, we are interested to applying our algorithms to a wider class of abduction and constraint optimization problems.

Acknowledgments

This work has been supported by STW grant DES.7015 and SFI grant 04/IN3/I524.

References

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of boolean cardinality constraints. In *CP*, 108–122.

Bylander, T.; Allemang, D.; Tanner, M.; and Josephson, J. 1991. The computational complexity of abduction. *Artificial Intelligence* 49:25–60.

de Kleer, J., and Kurien, J. 2003. Fundamentals of model-based diagnosis. In *Proc. Safeprocess 03*, 25–36.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3):197–222.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM* 42(1):3–42.

Feldman, A., and van Gemund, A. 2006. A two-step hierarchical algorithm for model-based diagnosis. In *Proc. AAAI'06*.

Feldman, A.; Provan, G.; and van Gemund, A. 2007. Generating manifestations of max-fault min-cardinality diagnoses. In *Proc. DX'07*.

Forbus, K., and de Kleer, J. 1993. *Building Problem Solvers*. MIT Press.

Freuder, E. C.; Dechter, R.; Ginsberg, B.; Selman, B.; and Tsang, E. P. K. 1995. Systematic versus stochastic constraint satisfaction. In *Proc. IJCAI 95*, volume 2.

Friedrich, G.; Gottlob, G.; and Nejdl, W. 1990. Physical impossibility instead of fault models. In *Proc. AAAI*.

Kask, K., and Dechter, R. Stochastic local search for Bayesian networks. In *Proc. AISTAT'99*.

McAllester, D. 1990. Truth maintenance. In *Proc. AAAI'90*, volume 2, 1109–1116.

Mozetič, I. 1992. A polynomial-time algorithm for model-based diagnosis. In *Proc. ECAI'92*, 729–733.

Provan, G. 2005. Approximate model-based diagnosis using preference-based compilation. In *Abstraction, Reformulation and Approximation*, volume 3607. Springer Berlin / Heidelberg. 182–193.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Vatan, F.; Barrett, A.; James, M.; Williams, C.; and Mackey, R. 2003. A novel model-based diagnosis engine: theory and applications. In *IEEE Aerospace Conference*.

Williams, B., and Ragno, R. 2004. Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*.

Zabih, R., and McAllester, D. 1988. A rearrangement search strategy for determining propositional satisfiability. In *Proc. AAAI'88*, 155–160.

Monitoring Plan Optimality during Execution: Theory and Implementation

Christian Fritz and **Sheila A. McIlraith**
Department of Computer Science, University of Toronto, Toronto, Ontario. Canada.
{fritz,sheila}@cs.toronto.edu

Abstract

A great deal of research has addressed the problem of generating optimal plans, but these plans are of limited use in circumstances where noisy sensors, unanticipated exogenous actions, or imperfect models result in discrepancies between predicted and observed states of the world during plan execution. Such discrepancies bring into question the continued optimality of the plan being executed and, according to current-day practice, are resolved by aborting the plan and replanning, often unnecessarily. In this paper we address the problem of monitoring the continued optimality of a given plan at execution time, in the face of such discrepancies. While replanning cannot be avoided when critical aspects of the environment change, our objective is to avoid replanning unnecessarily. We address the problem by building on practical approaches to monitoring plan *validity*. We begin by formalizing plan validity in the situation calculus and characterizing common approaches to monitoring plan validity. We then generalize this characterization to the notion of plan optimality and propose an algorithm that verifies continued plan optimality. We have implemented our algorithm and tested it on simulated execution failures in well-known planning domains. Experimental results yield a significant speed-up in performance over the alternative of replanning, clearly demonstrating the merit of our approach.

1 Introduction

When executing plans, the world may evolve differently than predicted resulting in discrepancies between predicted and observed states of the world. These discrepancies can be caused by noisy sensors, unanticipated exogenous actions, or by inaccuracies in the predictive model used to generate the plan in the first place. Regardless of the cause, when a discrepancy is detected, it brings into question whether the plan being executed remains *valid* (i.e., projected to reach the goal) and where relevant, *optimal* with respect to some prescribed metric. The task of execution monitoring is to monitor the execution of a plan, identify relevant discrepancies, and to take ameliorative action. In many cases the ameliorative action is to replan starting in the current state.

Effective execution monitoring requires a system to quickly discern between cases where a detected discrepancy is relevant to the successful execution of a plan and those cases where it is not. Algorithms dating back as far as 1972 (e.g., PLANEX (Fikes, Hart, & Nilsson 1972)) have exploited the idea of annotating plans with conditions that can be checked at execution time to confirm the continued *validity* of a sequential plan.

Here, we are interested in the more difficult and unsolved problem of monitoring plan *optimality*. Our work is motivated in part by our practical experience with the fast-paced RoboCup domain where teams of robots play soccer against each other. In RoboCup, the state of the world is typically observed 10 times per second, each time raising the question of whether to continue with the current plan or to replan. Verifying plan validity and optimality must be done quickly because of the rapidly changing environment. Currently, there are no techniques to distinguish between relevant and irrelevant discrepancies (w.r.t. optimality), and so replanning is frequently done unnecessarily or discrepancies are ignored altogether, ultimately resulting in plan failure or sub-optimal performance.

In this paper, we study the problem of monitoring the continued optimality of a plan. Our approach builds on ideas exploited in algorithms for monitoring plan validity. To this end, we begin by formalizing plan validity in the situation calculus, characterizing common approaches to monitoring plan validity found in the literature. We then generalize this characterization to the notion of plan optimality and propose an algorithm to monitor plan optimality at execution. Prior to execution time we annotate each step of our optimal plan by sufficient conditions for the optimality of the plan. These conditions correspond to the regression (Reiter 2001) of the evaluation function (cost + heuristic) used in planning over each alternative to the currently optimal plan. At execution time, when a discrepancy occurs, these conditions can be reevaluated much faster than replanning from scratch by exploiting knowledge about the specific discrepancy.

We have implemented our algorithm and tested it on simulated execution failures in well-known planning domains. Experimental results yield an average speed-up in performance of two orders of magnitude over the alternative of replanning, clearly demonstrating the feasibility and benefit of the approach. Further, while our approach is described in the situation calculus, it is amenable to use with any action description language for which regression can be defined (e.g., STRIPS and ADL).

The work presented in this paper does not investigate the problem of diagnosis proper, but is central to the broader

charge of diagnostic problem solving – detection of a discrepancy between predicted and observed state with the objective of determining ameliorative actions based on the discrepancy and its possible causes. Rather than focusing on discrepancy detection and diagnosis, we focus on determining what specific discrepancies are relevant to the successful execution of a system at each step of its plan or controller's execution. Knowing what is relevant to guarantee the continued validity or optimality of a plan or controller leads to focusing of the diagnosis effort on only the (small) subset of system state that is germane to the continued operation of the system. It also enables focused sensing or testing to discriminate diagnoses that are relevant to the continued operation of the system.

2 Preliminaries

The situation calculus is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents, $\mathcal{F}$) relativized to a particular *situation* s, e.g., $F(\vec{x}, s)$. A situation s is a *history* of the primitive actions a, $a \in \mathcal{A}$ the set of all actions, performed from a distinguished initial situation S_0. The function $do(a, s)$ maps an action and a situation into a new situation thus inducing a tree of situations rooted in S_0. For readability, action and fluent arguments are generally suppressed. Also, $do(a_n, do(a_{n-1}, \ldots do(a_1, s)))$ is abbreviated to $do([a_1, \ldots, a_n], s)$ or $do(\vec{a}, s)$. In this paper we only consider finite sets of actions, $\mathcal{A}$.

A basic action theory in the situation calculus, $\mathcal{D}$, comprises four *domain-independent foundational axioms*, and a set of *domain-dependent axioms*. Details of the form of these axioms can be found in (Reiter 2001). We write $s \sqsubset s'$ to say that situation s precedes s' in the tree of situations. This is axiomatized in the foundational axioms. Included in the domain-dependent axioms are the following sets of axioms.

Initial State, S_0: a set of first-order sentences relativized to situation S_0, specifying what is true in the initial state.

Successor state axioms: provide a parsimonious representation of frame and effect axioms under an assumption of the completeness of the axiomatization. There is one successor state axiom for each fluent, F, of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among $a, s, \vec{x}$. $\Phi_F(\vec{x}, a, s)$ characterizes the truth value of the fluent F in the situation $do(a, s)$ in terms of what is true in the current situation s.

Action precondition axioms: specify the conditions under which an action is possible to execute. There is one axiom for each action A, of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ where $\Pi_A(\vec{x}, s)$ is a formula with free variables among $\vec{x}, s$. We use the abbreviation $Poss([a_1, a_2, \ldots, a_m], s) \stackrel{\text{def}}{=} Poss(a_1, s) \wedge Poss(a_2, do(a_1, s)) \wedge \ldots \wedge Poss(a_m, do([a_1, \ldots, a_{m-1}], s))$.

Regression

The *regression* of a formula ψ through an action a is a formula ψ' that holds prior to a being performed if and only if ψ holds after a is performed. In the situation calculus, regression is defined inductively using the successor state axiom for F as above:

$$\mathcal{R}[F(\vec{x}, do(a, s))] = \Phi_F(\vec{x}, a, s)$$
$$\mathcal{R}[\neg\psi] = \neg\mathcal{R}[\psi]$$
$$\mathcal{R}[\psi_1 \wedge \psi_2] = \mathcal{R}[\psi_1] \wedge \mathcal{R}[\psi_2]$$
$$\mathcal{R}[(\exists x)\psi] = (\exists x)\mathcal{R}[\psi]$$

We denote the repeated regression of a formula $\psi(do(\vec{a}, s))$ back to a particular situation s by $\mathcal{R}_s$, e.g. $\mathcal{R}_s[\psi(do([a_1, a_2], s))] = \mathcal{R}[\mathcal{R}[\psi(do([a_1, a_2], s))]]$.

Intuitively, the regression of a formula ψ over an action sequence $\vec{a}$ is the condition that has to hold for ψ to hold after executing $\vec{a}$. It is predominantly comprised of the fluents that play a role in the conditional effects of the actions in the sequence.

Regression is a purely syntactic operation. Nevertheless, it is often beneficial to simplify the resulting formula for later evaluation. Regression can be defined in many action specification languages. In STRIPS, regression of a literal l over an action a is defined based on the add and delete lists of a: $\mathcal{R}^{\text{STRIPS}}[l] = \text{FALSE}$ if $l \in \text{DEL}(a)$ and $\{l\} \setminus \text{ADD}(a)$ otherwise. Regression in ADL was defined in (Pednault 1989).

Notation: Lower case letters denote variables in the theory of the situation calculus, upper case letters denote constants. We use α and β to denote arbitrary but explicit actions. However, we use capital S to denote arbitrary but explicit situation terms, that is $S = do(\vec{\alpha}, S_0)$ for some explicit action sequence $\vec{\alpha}$. For instance, S_i for $i > 0$ denotes the situation expected during planning, and S^* the actual situation that arises during execution. Variables that appear free are implicitly universally quantified unless stated otherwise and $\varphi[x/y]$ denotes the substitution of all occurrences of x in formula φ with y

3 Monitoring Plan Validity

In this section we formalize the notion of plan validity and provide an algorithm for monitoring plan validity. This provides the formal foundation for our approach to monitoring plan optimality described in Section 4.

Recall that a situation is simply a history of actions executed starting in S_0, e.g., $do([\alpha_1, \ldots, \alpha_m], S_0)$.

Definition 1 (Plan Validity)**.** Given a basic action theory $\mathcal{D}$ and a goal formula $G(s)$, a plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$ is *valid* in situation S if $\mathcal{D} \models G(do(\vec{\alpha}, S)) \wedge Poss(\vec{\alpha}, S)$.

As such, a plan continues to be valid if, according to the action theory and the current situation, the precondition of every action in the plan will be satisfied, and at the end of plan execution, the goal is achieved.

A number of systems have been developed for monitoring plan validity (cf. Section 6) which all implicitly take the following similar approach. The planner annotates each step of the plan with a sufficient and necessary condition that confirms the validity of the plan. During plan execution these conditions are checked to determine whether plan execution should continue. We formally characterize the annotation and its semantics as goal regression. The provision of such a characterization enables its exploitation with other planners, such as very effective heuristic forward search planners.

Definition 2 (Annotated Plan)**.** Given initial situation S_0, a sequential plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, and a goal formula $G(s)$, the corresponding annotated plan for $\vec{\alpha}$ is a sequence of tuples $\pi(\vec{\alpha}) = (G_1(s), \alpha_1), (G_2(s), \alpha_2), \ldots, (G_m(s), \alpha_m)$ such that

$$G_i(s) = \mathcal{R}_s\left[G(do([\alpha_i, \ldots, \alpha_m], s) \wedge \mathit{Poss}([\alpha_i, \ldots, \alpha_m], s)\right]$$

I.e., each step is annotated with the regression of the goal and the preconditions over the remainder of the plan.

Proposition 1. A sequence of actions $\vec{\alpha}$ is a valid plan in situation S iff $\mathcal{D} \models \mathcal{R}_S[G(do(\vec{\alpha}, S)) \wedge \mathit{Poss}(\vec{\alpha}, S)]$.
Proof: The proof is by induction using the Regression Theorem (Reiter 2001, pp.65–66).

We can now provide an algorithm that characterizes the approach to monitoring plan validity described above. It is a generalization of the algorithm defined in (Fikes, Hart, & Nilsson 1972). For the purposes of this paper, we assume that the "actual" situation of the world, S^*, is provided, having perhaps been generated using state estimation techniques or the like. The action theory $\mathcal{D}$ remains unchanged. For instance, S^* may differ from the expected situation $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$ by containing unanticipated exogenous actions that happened, or variations of actions executed by the agent. It need not provide a *complete* description of the state of the world. The approach is applicable and even particularly interesting in cases of incomplete knowledge. For a particular example of how to conjecture the actual situation S^* in the situation calculus, see (McIlraith 1998; Iwan 2000). The idea is, roughly, to conjecture sequences of exogenous actions or variations of executed agent actions such that the resulting sequence explains the observation. This can be interpreted as a planning problem where the goal is defined to be the observation.

Definition 3 (Algorithm for Monitoring Plan Validity)**.** With action theory $\mathcal{D}$ and annotated plan $\pi(\vec{\alpha})$ of length $\mathbf{m}$[1]

obtain S^*
while $(\mathcal{D} \not\models G(S^*))$ {
  $\mathbf{i} = \mathbf{m}$; *obtain* S^*
  while $(\mathcal{D} \not\models G_{\mathbf{i}}(S^*))$ { $\mathbf{i} = \mathbf{i} - 1$}
  if $(\mathbf{i} > 0)$ **then** $execute\ \alpha_{\mathbf{i}}$ **else** $replan$ }

The realization of the entailment of conditions ($\mathcal{D} \models \varphi(s)$) depends on the implemented action language. E.g., in STRIPS this is simple set inclusion of literals in the set describing the current state (roughly, $\varphi \in s$). For the situation calculus efficient Prolog implementations exist.

The correctness of this approach – only valid plans are continued and whenever a plan is still valid it is continued – is provided by the following theorem.

Theorem 1. The algorithm executes action α_i in situation S^* iff the remaining plan $[\alpha_i, \ldots, \alpha_m]$ is valid in situation S^* and i is the greatest index in $[1, m]$ with that property.
Proof: If α_i is executed, $\mathcal{D} \models G_i(S^*)$ holds and plan validity follows by Propositions 1. If there is a maximal index i such that $\mathcal{D} \models G_i(S^*)$ (plan valid by Prop. 1), α_i is executed.

[1]To better distinguish variables of the theory and meta-variables, used e.g. in pseudo code, we print the latter using bold face.

4 Monitoring Plan Optimality

Now that we have a formal understanding of what is required to monitor plan validity, we exploit this to address the challenging problem of monitoring the continued optimality of a plan. Optimality appears in cases where the user not only specifies a goal to define valid plans, but also wishes to discriminate between all possible valid plans, by designating a measure of utility or preference over plans.

Given an optimal plan, our objective is to monitor its continued optimality, electing to replan only in those cases where continued execution of the plan will either not achieve the goal, or will do so sub-optimally. To this end, we extend the plan-annotation approach of Section 3 to monitor plan optimality. This is a critical problem for many real-world planning systems, and one that has been largely ignored.

We begin by defining plan optimality within our framework. Recall that $do(\vec{\alpha}, S)$ denotes the situation reached after performing the action sequence $\vec{\alpha}$ in situation S. The relation $\mathit{Pref}(s, s')$ is an abbreviation for a sentence in the situation calculus defining criteria under which s is more or equally preferred to s'.

Definition 4 (Plan Optimality)**.** Given a basic action theory $\mathcal{D}$, a goal formula $G(s)$, and extralogical relation $\mathit{Pref}(s, s')$, a plan $\vec{\alpha}$ is *optimal* in situation S if $\mathcal{D} \models G(do(\vec{\alpha}, S)) \wedge \mathit{Poss}(\vec{\alpha}, S)$ and there is no action sequence $\vec{\beta}$ such that $\mathcal{D} \models \mathit{Pref}(do(\vec{\beta}, S), do(\vec{\alpha}, S)) \wedge G(do(\vec{\beta}, S)) \wedge \mathit{Poss}(\vec{\beta}, S)$.

As such, a plan remains optimal in a new situation S^* when it remains valid and there exists no other valid plan that is preferred. Hence, to monitor plan optimality, we require two changes to our plan annotations: i) in addition to regressing the goal, we must regress the preference criteria to identify conditions that are necessary to enforce optimality and ii) since optimality is relative rather than absolute, we must annotate each plan step with the regression of the preferences over alternative plans as well.

This approach can be applied to a wide range of preference representation languages and planners. In this paper, we restrict our attention to preferences described by positive numeric action costs, and an A^* search based forward planner with an admissible evaluation function. To provide a formal characterization, we assume that the planning domain is encoded in a basic action theory $\mathcal{D}$. To keep the presentation simple, we also assume that the goal is a fluent $G(s)$ and can only be established by a particular action finish, contained in the action theory. Any planning problem can naturally be translated to conform to this by defining the preconditions of the *finish* action corresponding to the goal of the original planning problem.

Recall that in A^* search, the evaluation function is the sum of the heuristic function and the accumulated action costs. We denote functions in relational form. In situation s the accumulated cost c of actions performed since S_0 is denoted by the relational fluent $\mathit{Cost}(c, s)$. It is specified incrementally in the successor state axioms of the fluent $\mathit{Cost}(c, s)$. For instance, $\mathit{Cost}(c, do(a, s)) \equiv \mathit{Cost}(c', s) \wedge (a = driveTo(paris) \wedge c = c' + driveCostToParis) \vee ((\nexists x).a = driveTo(x) \wedge c = c')$. The search heuristic yielding value h in s is denoted by $\mathit{Heu}(h, s)$. We understand this to denote a formula provided by the user, for instance of the form $\mathit{Heu}(h, s) \stackrel{\text{def}}{=} (\phi_1(s) \wedge h =$

$h_1) \vee (\phi_2(s) \wedge h = h_2) \vee \cdots \vee (\phi_n(s) \wedge h = h_n)$, where the ϕ_i partition state space. Correspondingly our A^* evaluation relation is specified through the following formula:

$$Value(v, s) \stackrel{\text{def}}{=} (\exists h, c).Heu(h, s) \wedge Cost(c, s) \wedge v = h + c.$$

The preference relation for our A^* search is defined as:

$$Pref_{A^*}(s_1, s_2) \stackrel{\text{def}}{=} (\exists v_1, v_2).\, Value(v_1, s_1) \wedge Value(v_2, s_2) \wedge v_1 \leq v_2.$$

By the definition of admissibility we know that it is non-decreasing, that is if s_1 is preferred to s_2, then no successor of s_2 is preferred to s_1:

$$\mathcal{D} \models Pref_{A^*}(s_1, s_2) \supset (\nexists s_2').s_2 \sqsubset s_2' \wedge Pref_{A^*}(s_2', s_1). \quad (1)$$

In A^* search, nodes that have been seen but not explored are kept in the so-called *open list*. In planning, the open list is initialized to the initial situation. Planning proceeds by repeatedly removing the most preferred situation in the open list and inserting its feasible successor situations. Search terminates successfully when this first element, say $do(\vec{\alpha}, S_0)$, satisfies the goal, $\mathcal{D} \models G(do(\vec{\alpha}, S_0))$. The plan described by this, $\vec{\alpha}$, is always optimal because any alternative plan would be a continuation of one of the partial plans in the open list, but by Equation (1) and the fact that $do(\vec{\alpha}, S_0)$ is preferred to any other element in the open list, no such plan could be preferred to $\vec{\alpha}$.

It follows that to determine the continued optimality of plan $\vec{\alpha}$ in a new situation S^* (replacing S_0), it is *sufficient* to check that $do(\vec{\alpha}, S^*)$ is still most preferred with respect to the open list when search terminated. We may, however, need to extend this list first because action sequences previously predicted to be impossible (in S_i) may now be possible (in S^*) and the current plan, $\vec{\alpha}$, must also be preferred to these plans. But even then this is not a *necessary* condition. It may be the case that another element $do(\vec{\beta}, S^*)$ in the (revised) open list is preferred to $do(\vec{\alpha}, S^*)$, but if $do(\vec{\beta}, S^*)$ doesn't satisfy the goal, $do(\vec{\alpha}, S^*)$ may still turn out to be optimal. This could be the case if (i) no successor of $do(\vec{\beta}, S^*)$ satisfies the goal, or (ii) there are successors that satisfy the goal but they are less preferred than $do(\vec{\alpha}, S^*)$. This can occur because the heuristic can, and generally does, increase. Formally, $\mathcal{D} \models Pref_{A^*}(s_1, s_2) \not\supset (\nexists s_1').s_1 \sqsubset s_1' \wedge Pref_{A^*}(s_2, s_1')$. Neither of these issues, can be resolved without further, time consuming, planning. For this reason, we limit ourselves to a tight sufficient condition, defined in terms of the fringe.

Definition 5 (Fringe)**.** Given an action theory $\mathcal{D}$ and a goal formula $G(s)$, a *fringe* of situation S is a set of situations $L = \{S_1, \ldots, S_n\}$, such that for each $S_j \in L$: (i) S_j is a descendant of S, (i.e., $S \sqsubset S_j$), (ii) there is no $S' \in L$ s.t. $S_j \sqsubset S'$, (iii) for any S'' if $do(\alpha, S'') \in L$ for some $\alpha \in \mathcal{A}$ then also $do(\alpha', S'') \in L$ for every other $\alpha' \in \mathcal{A}$, where $\mathcal{A}$ is the set of actions in the theory, and (iv) there is no S''', $S''' \sqsubset S_j$ such that $\mathcal{D} \models G(S''')$.

Note the similarity between fringe and open list. The main difference is that a fringe can include infeasible situations.

Theorem 2 (Sufficiency)**.** Let $\mathcal{D}$ be an action theory, $G(s)$ a goal, S a situation, and $\vec{\alpha}$ a valid plan for $G(s)$ in S.

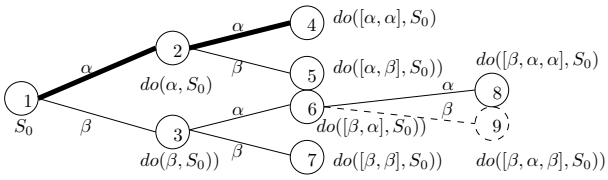


Figure 1: An example search tree. Dashed lines denote impossible actions, and $[\alpha, \alpha]$ is the optimal plan.

If there is a fringe L of S such that for every $do(\vec{\beta}, S) \in L$, $\mathcal{D} \models (\exists v_a, v_b).\mathcal{R}_S[Poss(\vec{\beta}, S) \supset Value(v_a, do(\vec{\alpha}, S)) \wedge Value(v_b, do(\vec{\beta}, S))] \wedge v_b \geq v_a$, then $\vec{\alpha}$ is optimal in S.[2]

Proof: Assume to the contrary that $\vec{\alpha}$ is not optimal in S, then there is a $s_b = do(\vec{\beta}, S)$ s.t. $\mathcal{D} \models (\exists v_a, v_b).Value(v_a, do(\vec{\alpha}, S)) \wedge Value(v_b, s_b) \wedge v_b < v_a$ and $\mathcal{D} \models G(s_b) \wedge Poss(\vec{\beta}, S)$. By definition of a fringe, s_b is either in L or is a predecessor or a descendant of some element in L. Since no element in L is preferred to $\vec{\alpha}$ and no predecessor of any element in L satisfies the goal, s_b has to be a descendant of some element in L. But by definition of admissibility (Equation (1)), no such descendant can be preferred to $\vec{\alpha}$. □

This theorem establishes sufficient conditions for determining continued plan optimality. We must now translate these conditions into plan annotations that can be quickly checked during plan execution.

4.1 Annotation

Plan annotations can be computed at planning time by our A^* search forward planner. We assume our planner will output an optimal plan $\vec{\alpha}$, the open list O that remained when planning terminated (e.g. nodes 5, 7, and 8 in Figure 1), and a list O^- containing those situation terms found to be infeasible during planning. This is a list of situations $do(\alpha^-, S)$ such that α^- is not possible in situation S, i.e. $\mathcal{D} \models \neg Poss(\alpha^-, S)$ (cf. node 9 in the figure). The union $\{do(\vec{\alpha}, S_0)\} \cup O \cup O^-$ is a fringe of S_0. Since monitoring optimality must be done relative to alternatives, each step of the optimal plan is annotated with the conditions that confirm its continued validity, as well as a list of alternative plans and their corresponding predicted evaluation function values relativized to that plan step. Note that in the text that follows it is the description of the annotation, and not the annotation itself, that is defined in the situation calculus.

Definition 6 (Annotated Plan (Optimality))**.** Given the initial situation S_0, the goal formula $G(s)$, the evaluation relation $Value(v, s)$, an optimal sequential plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, the open list O, and the list of infeasible situations O^-, the corresponding annotated plan for $\vec{\alpha}$ is a sequence of tuples

$$\pi(\vec{\alpha}) = (G_1(s), V_1, Alt_1, \alpha_1), \ldots, (G_m(s), V_m, Alt_m, \alpha_m)$$

where $G_i(s)$ is as defined in Definition 2 and V_i and Alt_i are defined as follows, with $\vec{\alpha}_i = [\alpha_i, \ldots, \alpha_m]$ the remaining plan, and $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$:

$V_i = (\mathcal{R}_s[Value(v, do(\vec{\alpha}_i, s))], \mathbf{v_i})$,

[2] Since $\vec{\beta}$ is a particular, known action sequence, regressing over it is not a problem and our abbreviation $Poss(\vec{\beta}, S)$ is well defined.

with $\mathbf{v_i} \in I\!R$ such that $\mathcal{D} \models Value(\mathbf{v_i}, do(\vec{\alpha}_i, S_i))$
$Alt_i = \{Alt_{i_1}, Alt_{i_2}, \ldots, Alt_{i_n}\}$ containing all tuples
$Alt_{i_j} = (\vec{\beta}_{i_j}, \phi_{i_j}^{Poss}, \mathbf{p_{i_j}}, \phi_{i_j}^{Value}, \mathbf{v_{i_j}})$
such that $do(\vec{\beta}_{i_j}, S_i) \in O \cup O^-$ and where:
$\phi_{i_j}^{Poss} = \mathcal{R}_s[Poss(\vec{\beta}_{i_j}, s)]$, variable in s,
$\mathbf{p_{i_j}} = 1$ if $do(\vec{\beta}_{i_j}, S_i) \in O$ and $\mathbf{p_{i_j}} = 0$ if $do(\vec{\beta}_{i_j}, S_i) \in O^-$,
$\phi_{i_j}^{Value} = \mathcal{R}_s[Value(v, do(\vec{\beta}_{i_j}, s)]$, variable in v, s, and
$\mathbf{v_{i_j}} \in I\!R$ such that $\mathcal{D} \models Value(\mathbf{v_{i_j}}, do(\vec{\beta}_{i_j}, S_i))$.

For plan step i, V_i contains the regression of the evaluation relation over the remaining plan $\vec{\alpha}_i$ and its value w.r.t. the expected situation S_i. Alt_i represents the list of alternative action sequences $\vec{\beta}_{i_j}$ to $\vec{\alpha}_i$, together with their respective regression of the evaluation relation and particular value in S_i ($\mathbf{v_{i_j}}$), and regressed preconditions and their truth value ($\mathbf{p_{i_j}}$, represented as 0 or 1) in S_i. For example, in the search tree of Figure 1, $\vec{\alpha} = [\alpha, \alpha]$ is the optimal plan, Alt_1 (cf. node 1) contains tuples for the action sequences $[\alpha, \beta], [\beta, \alpha, \alpha], [\beta, \alpha, \beta], [\beta, \beta]$, and Alt_2 (cf. node 2) contains only one tuple for $[\beta]$. Intuitively, the regression of the evaluation relation over a sequence of actions $\vec{\alpha}$ describes in terms of the current situation, the value the evaluation relation will take after performing $\vec{\alpha}$. As an example, consider the task of delivering a package to a location using a truck. Assume the heuristic yields a value $v = 0$ when the truck has the package loaded and is at the right location, and $v = 1$ otherwise. Then, regressing the heuristic through the action of driving the truck to the right location would yield a formula stating "$v = 0$ *if the package is on the truck, and* $v = 1$ *otherwise*" (ignoring action costs for now).

The *key* benefit of our approach comes from regressing conditions to the situations where they are relevant. Consequently when a discrepancy is detected during plan execution and the world is in situation S^* rather than predicted situation S_i, the monitor can determine the difference between these situations and limit computation to reevaluating those conditions that are affected by the discrepancy. This is only possible because regression has enabled definition of relevant conditions with respect to the situation before executing the remainder of the plan or any alternative.

4.2 Execution Monitoring

Assume we have an optimal plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, and that we have executed $[\alpha_1, \ldots, \alpha_{i-1}]$ for some $i \leq m$ and thus expect to be in situation $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$. Given the situation estimate S^* and the annotated plan as described in Definition 6, our task is to decide whether execution of the remainder of the plan, $\vec{\alpha}_i = [\alpha_i, \ldots, \alpha_m]$, is still optimal. We will do this by reevaluating all relevant conditions in S^* to verify that the current plan is still valid and achieves maximal value among all alternatives.

Recall the representation of alternative plans $\vec{\beta}$ in Alt_i, containing the regressed preconditions, evaluation relation, and their respective 'values' during planning (i.e. w.r.t. S_i). Also recall that $V_i = (\phi^{Value}, \mathbf{v_i})$ where ϕ^{Value} is a formula over variables v and s, denoting the regression of the evaluation relation over $\vec{\alpha}_i$. A naive algorithm for monitoring plan optimality at execution would be as follows:

Definition 7 (Naive Optimality Monitoring Algorithm)**.** Given the annotated plan $\pi(\vec{\alpha}) = (G_1(s), V_1, Alt_1, \alpha_1), \ldots, (G_m(s), V_m, Alt_m, \alpha_m)$:

```
i = 1
while (i ≤ m) {
  obtain S*
  (φ^Value, v_i) = V_i
  if (D ⊨ G_i(S*) and ∃v_a ∈ IR s.t. D ⊨ φ^Value[s/S*, v/v_a]
      and ∀(β, φ_β^Poss, p_β, φ_β^Value, v_β) ∈ Alt_i, ∃v_b ∈ IR s.t.
      D ⊨ φ_β^Poss[s/S*] ∧ φ_β^Value[s/S*, v/v_b] ∧ v_b ≥ v_a)
  then { execute α_i; i = i + 1; }
  else replan }
```

This prescribes to continue execution as long as no feasible element from the list of alternatives achieves a better value in S^* than the current plan. The time cost of this algorithm is greatly determined by the computation of the condition as it reevaluates all annotated formulae anew in S^*. We can significantly reduce this time by only reevaluating those conditions that may have been affected by the discrepancy between the predicted situation S_i and actual situation S^*.

Let $\Delta_F(S_i, S^*)$ be the set of fluents whose truth values differ between S_i and S^*, i.e. $\Delta_F(S_i, S^*) = \{F(\vec{X}) \mid F \in \mathcal{F}$ and $\mathcal{D} \models F(\vec{X}, S_i) \not\equiv F(\vec{X}, S^*)\}$, with $\mathcal{F}$ the set of fluents. Only conditions mentioning any of these fluents need to be reevaluated, all others remain unaffected by the discrepancy. Let *fluents*(φ) denote the set of all fluents occurring in φ. An improved algorithm for monitoring plan optimality during execution is as follows:

Definition 8 (Monoplex)**.** Given the annotated plan $\pi(\vec{\alpha}) = (G_1(s), V_1, Alt_1, \alpha_1), \ldots, (G_m(s), V_m, Alt_m, \alpha_m)$
$monoplex(\mathcal{D}, \pi(\vec{\alpha}))$:

```
1  i = 1
2  while (i ≤ m) {
3    obtain S*; generate Δ_F(S_i, S*);
4    if (D ⊨ G_i(S*)) then {        // plan remains valid
5      (φ^Value, v_i) = V_i
6      if (fluents(φ^Value) ∩ Δ_F(S_i, S*) ≠ ∅) then
7        { v_i = v_i^new s.t. D ⊨ φ^Value[s/S*, v/v_i^new] }
8      foreach ((β, φ_β^Poss, p_β, φ_β^Value, v_β) ∈ Alt_i) {
9        if (fluents(φ_β^Poss) ∩ Δ_F(S_i, S*) ≠ ∅) then
10         { if (D ⊨ φ_β^Poss[s/S*]) then p_β = 1 else p_β = 0 }
11       if (p_β == 1 ∧ fluents(φ_β^Value) ∩ Δ_F(S_i, S*) ≠ ∅) then
12         { v_β = v_β^new s.t. D ⊨ φ_β^Value[s/S*, v/v_β^new] }
13       if (p_β == 1 ∧ v_β < v_i) then
14         { replan } }        // plan may be sub-optimal, replan
15     execute α_i; i = i+1      // plan remains optimal, continue
17   else replan } }      // plan is invalid
```

While the plan has not been executed to completion (line 2), the algorithm does the following:

line 4: it checks validity;

lines 5–7: if the regression of the evaluation function over the plan (ϕ^{Value}) mentions any affected fluents, it is reevaluated, obtaining new value $\mathbf{v_i^{new}}$;

lines 8–14: the algorithm then checks for each alternative $\vec{\beta}$ at this point of plan execution: (in lines 9,10) whether

its preconditions are affected and need to be reevaluated, (in lines 11,12) whether its value is affected and needs to be reevaluated, and (in line 13) whether this alternative is now possible and better than the current plan. If an alternative has become better, the algorithm calls for replanning. Otherwise the next action of the plan is executed.

Intuitively, the **foreach** loop revises relevant values – the truth of preconditions and the value of the evaluation function – generated for S_i with respect to the actual situation S^*, aborting execution when a viable and superior alternative is found. Line 12 is most crucial: Here the regression of the evaluation relation over alternative plan $\vec{\beta}$ is reevaluated with respect to the actual current situation S^*, yielding a new value $\mathbf{v}_\beta^{\mathbf{new}}$ for the evaluation relation (cf. Alt_{i_j} in Definition 6). This reevaluation only occurs if the regression result (formula) contains any of the affected fluents. Otherwise the value hasn't changed as result of the discrepancy. Again, the realization of the entailment of conditions ($\mathcal{D} \models \varphi(s)$) depends on the implemented action language. The method is in particular not reliant on the situation calculus and can be used with any action language for which regression can be defined.

Theorem 3 (Correctness)**.** Whenever $monoplex$ executes the next step of the plan (*execute* α_i), the remaining plan $\alpha_i, \ldots, \alpha_m$ is optimal in the actual current situation S^*.

Exploiting the Search Tree Working with the fringe directly causes a lot of redundant work. This is because many alternative action sequences share the same prefix and so the costs and preconditions of these prefixes are annotated and potentially reevaluated multiple times. We can avoid this by exploiting the search tree structure in both the annotation and the algorithm. The details of this method are much more complicated than the above description based on the list of alternatives, which is why we chose to omit these details in this paper. Our implementation, however, uses the improved search tree method. Formally the search tree method is equivalent to the described method with respect to making the decision between continuing and aborting/replanning.

4.3 An Illustrative Example

Consider the following simplified example from the TPP domain, where an agent drives to various markets to purchase goods which she then brings to the depot. For simplicity, assume there is only one kind of good, two markets, and the following fluents: in situation s, $At(l, s)$ denotes the current location l, $Totalcost(t, s)$ denotes the accumulated costs t of all actions since S_0, $Request(r, s)$ represents the number r requested of the good, $Price(p, m, s)$ denotes the price p of the good on market m, and $DriveCost(c, src, dest, s)$ the cost c of driving from src to $dest$. Let there be two actions: $drive(dest)$ moves the agent from the current location to $dest$, and $buyAllNeeded$ purchases the requested number of goods. Assume, the planner has determined the plan $\vec{\alpha} = [drive(Market1), buyAllNeeded, drive(Depot)]$ to be optimal, but has as well considered $\vec{\beta} = [drive(Market2), buyAllNeeded, drive(Depot)]$ as one alternative among others. To shorten presentation, we ignore the heuristic here, i.e. assume uniform cost search ($h = 0$). Then

$$V_1 = (\ (\exists t, l, c_1, p, r, c_2).Totalcost(t, s) \wedge At(l, s) \wedge DriveCost(c_1, l, Market1, s) \wedge Price(p, Market1, s) \wedge Request(r, s) \wedge DriveCost(c_2, Market1, depot, s) \wedge v = t + c_1 + (r \cdot p) + c_2,\ \mathbf{v_1}\ )$$

and similarly $Alt_{1_1} = (\vec{\beta}, \phi_{1_1}^{Poss}, \mathbf{p_{1_1}}, \phi_{1_1}^{Value}, \mathbf{v_{1_1}})$ where, very similar to above, $\phi_{1_1}^{Value} = (\exists t, l, c_1, p, r, c_2).Totalcost(t, s) \wedge At(l, s) \wedge DriveCost(c_1, l, Market2, s) \wedge Price(p, Market2, s) \wedge Request(r, s) \wedge DriveCost(c_2, Market2, depot, s) \wedge v = t + c_1 + (r \cdot p) + c_2$ where we ignore preconditions for simplicity and $\mathbf{v_1}$ and $\mathbf{v_{1_1}}$ are the respective values of the cost function for the plan and the alternative with respect to the situation where we expect to execute this plan, S_0.

Let's assume that even before the execution of the plan begins, a discrepancy in the form of an exogenous action e happens, putting us in situation $S^* = do(e, S_0)$ instead of S_0. The question is, whether $\vec{\alpha}$ is still optimal and in particular still better than $\vec{\beta}$. This clearly depends on the effects of e. If e does not affect any of the fluents occurring in above annotated formulae, it can be ignored, the plan is guaranteed to remain optimal. This would, for instance, be the case when e represents the event of a price change on a market not considered, as that price would not find it's way into the regressed formulae which only mention relevant fluents.

But even when e affects a relevant fluent, replanning may not be necessary. Assume, for instance, that e represents the event of an increased demand, that is, increasing the value r of $Request(r, s)$, formally $\mathcal{D} \models Request(r, S^*) > Request(r, S_0)$. Then $\Delta_F(S_0, S^*) = \{Request(r)\}$ and we need to reevaluate the annotated conditions, as $\vec{\beta}$ may have become superior. This could be the case, for instance, if the drive cost to $Market1$ is lower than to $Market2$, but the price at this market is higher. Then, a higher demand may make $Market2$ favorable, as the drive cost is compensated more than before by the lower price. This can quickly be determined by reevaluating the annotated conditions in S^*, obtaining new values $\mathbf{v_1}$ for $\vec{\alpha}$ and $\mathbf{v_{1_1}}$ for $\vec{\beta}$. If $\mathbf{v_{1_1}} < \mathbf{v_1}$ we have to replan, otherwise the plan remains optimal.

5 Empirical Results

We have proven that our approach establishes conditions under which plan optimality persists in a situation. We were interested in determining whether the approach was time-effective – whether the discrepancy-based incremental reevaluation could indeed be done more quickly than simply replanning when a discrepancy was detected. As noted in the introduction, continuous replanning has already been determined to be ineffective in highly-dynamic domains.

To this end, we compared a preliminary implementation of our `monoplex` algorithm to replanning from scratch on 9 different problems in the metric `TPP` domain of the 5^{th} International Planning Competition. In each case, we solved the original planning problem, perturbed the state of the world by changing some fluents, and then ran both `monoplex` and replanning from scratch. To maximize objectivity, the perturbations were done systematically by mul-

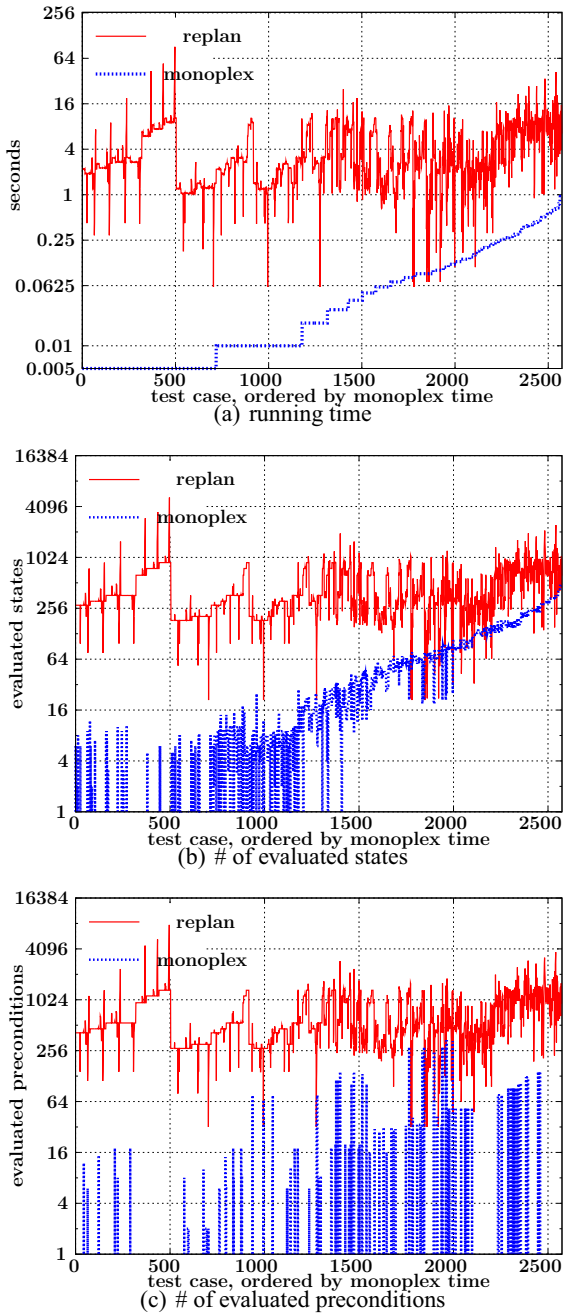


Figure 2: `TPP domain` (note the logarithmic scale)

tiplying the value of one of the numeric fluents by a factor between 0.5 and 1.5 (step-size 0.1), or by changing the truth value of a boolean fluent. This resulted in a total of 2574 unique test cases. Figure 2 shows the performance of both approaches on a logarithmic scale (all experiments were run on an Intel Xeon, 3.6GHz, 1GB RAM). To enhance readability we ordered the test cases by the running time of `monoplex`. The determining factors for the running time (cf. Figure 2a) are predominantly the number of states for which the evaluation function had to be reevaluated (2b), and the number of reevaluated action preconditions (2c). The results show that although it is possible for `monoplex` to be slower than replanning (in 8 out of 2574 cases), it generally performs much better, resulting in a pleasing average speed-up of 209.12. In 1785 cases the current plan was asserted still optimal and therefore replanning unnecessary, in 105 it had become invalid. In 340 of the remaining 684 cases, replanning found the current plan to still be optimal. Notice in (b) and (c) that sometimes the reevaluation of states and/or preconditions can be entirely avoided, namely when the perturbation does not affect any relevant fluents. This happened 545 times and constitutes the greatest time savings potential, a result of our formal characterization of the situation-dependent relevance of fluents to the optimality of the plan.

We obtained similar results on problems of the open stacks domain.

6 Related Work

The use of some form of plan annotation to monitor the continued validity of a plan during execution has been exploited in a number of systems (e.g., (Fikes, Hart, & Nilsson 1972; Wilkins 1985; Ambros-Ingerson & Steel 1988; Kambhampati 1990)), however none identified the formal foundations of the annotations as regression of the goal to relevant plan steps. Annotations (implicitly the regression) were computed as part of the planning algorithm (backward chaining, POP, or HTN planning). Our formalization in Section 3 elucidates this approach making it easily applicable to other planning algorithms.

To the best of our knowledge, the SHERPA system (Koenig, Furcy, & Bauer 2002) is the sole previous work that addresses the problem of monitoring the continued optimality of a plan, though only in a limited form. SHERPA lifts the Life-Long Planning A^* (LPA^*) search algorithm to symbolic propositional planning. LPA^* was developed for the purpose of replanning in problems like robot navigation (i.e. path-(re-)planning). As such this algorithm only applies to replanning problems where the current state remains unchanged, but the costs of actions in the search tree have changed. This is a major limitation. Similar to our approach, SHERPA retains the search tree to determine how changes may affect the current plan. But again, these changes are limited to action costs, while our approach guarantees optimality in the general case.

Another advantage of our approach is that it facilitates active sensing on the part of the agent. Our annotations can be exploited by the agent to quickly discern conditions that are relevant to a situation. Others have recognized the importance of this issue (cf. e.g. (Doyle, Atkinson, & Doshi 1986)), but to the best of our knowledge we are the first to address it with respect to the relevant features for optimality, rather than just for validity.

Also related is (Veloso, Pollack, & Cox 1998) in which the authors exploit the 'rationale', the reasons for choices made during planning, to deal with discrepancies that occur during *planning*. The authors acknowledge the possibility that previously sub-optimal alternatives become better than the current plan candidate as the world evolves during planning, but the treatment of optimality is informal and limited.

In (De Giacomo, Reiter, & Soutchanski 1998) the authors formalize an execution monitoring framework in the situation calculus to monitor the execution of Golog programs. However, they too are only concerned with plan validity. They do not follow the approach of goal regression but instead use projection every time a discrepancy is detected.

Also worth noting is the distinction between our approach to execution monitoring and so-called "Universal Plans" (Schoppers 1987). Roughly, universal plans perform replanning ahead of time, by regressing the goal over all possible action sequences. This approach is infeasible, as it grows exponentially in the number of domain features. The complexity of our approach on the other hand is bounded by the size of the search tree expanded during planning.

Also related are the results of (Nebel & Koehler 1995). They show that plan reuse – and plan repair in reaction to unforeseen discrepancies is a special case of this – is as complex as planning from scratch in worst case, and that this result even holds for cases where the considered planning instances are very similar. These results further motivate our work, as they highlight the benefit of avoiding replanning entirely whenever possible.

7 Summary and Future Work

When executing plans in dynamic environments, discrepancies between the expected and actual state of the world can arise for a variety of reasons. When such circumstances cannot be anticipated and accounted for during planning, they bring into question whether they are relevant, whether they render the current plan invalid or sub-optimal. While there are seemingly several approaches for monitoring validity, no approaches exist for monitoring optimality. Instead it is common practice to replan when a discrepancy occurs or to ignore the discrepancy, accepting potentially suboptimal behavior. Time-consuming replanning is impractical in highly dynamic domains and many discrepancies are irrelevant and thus replanning unnecessary, but to maintain optimality we have to determine which these are.

This paper makes several contributions to this problem. We provided a formal characterization of a common technique for monitoring plan validity based on annotating the plan with conditions for the continued validity, which are checked during execution. We then generalized this to the notion of plan optimality, providing a sufficient condition for optimality and an algorithm that exploits knowledge about the actual discrepancy to quickly test this condition at execution. This approach guarantees plan optimality while minimizing unnecessary replanning. The monitoring task includes the diagnostic problem of estimating the actual system state. Given the annotation technique proposed, the diagnosis effort can be focused on those aspects of system state that are relevant to the greater objective of verifying the validity/optimality of the plan. They also suggest focused sensing or testing, where required.

We implemented our algorithm and tested it on systematically generated execution discrepancies in the `TPP` and `open stacks` domains. The results show that many discrepancies are irrelevant, leaving the current plan optimal, and that our approach is much faster than replanning, with an average speed-up of two orders of magnitude.

In future work we plan to further investigate the nature of diagnosis within our proposed execution monitoring framework. The strategy of focusing discrepancy detection on those aspects of system state that affect plan validity/optimality suggests a slightly different approach to diagnosis. Instead of seeking the most likely diagnosis, we are interested first and foremost in the likelihood of being in a system state which preserves the validity/optimality of the plan. Finally, we are trying to find complexity results for deriving and testing necessary (not just sufficient) conditions, but continue to believe that the evaluation of such a condition would not outperform replanning.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI)*, 83–88.

De Giacomo, G.; Reiter, R.; and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 453–465.

Doyle, R.; Atkinson, D.; and Doshi, R. 1986. Generating perception requests and expectations to verify the execution of plans. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI'86)*, 81–88.

Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Iwan, G. 2000. Explaining what went wrong in dynamic domains. In *Proc. of the 2nd International Cognitive Robotics Workshop*.

Kambhampati, S. 1990. A theory of plan modification. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, 176–182.

Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic search-based replanning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS'02)*, 294–301.

McIlraith, S. 1998. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 167–177.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence (Special Issue on Planning and Scheduling)* 76(1-2):427–454.

Pednault, E. 1989. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, 324–332.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for continuous planning in dynamic environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, 171–179.

Wilkins, D. E. 1985. Recovering from execution errors in SIPE. *Computational Intelligence* 1:33–45.

Real World Model-based Fault Management

Ravi Kapadia
General Atomics, 16969 Mesamint Street, San Diego, CA 92127
Greg Stanley
Greg Stanley and Associates, 8619 Tranquil Park Drive, Spring, TX 77379
Mark Walker
General Atomics, 16969 Mesamint Street, San Diego, CA 92127

Abstract: Real world fault management applications encompass a number of diagnostic activities such as symptom monitoring, root cause analysis, impact prediction, testing, and recovery. They motivate powerful knowledge representation schemes to capture domain expertise and the development of intelligent algorithms that can exploit this knowledge. There are vast opportunities for the application of state-of-the-art fault management in commercial settings and, with billions of dollars at stake, industries are eager to embrace intelligent knowledge based solutions. Over the past decade, we have developed an object-oriented model-based domain-independent methodology for real world fault management, called SymCure. In this paper, we use this experience to generalize a set of requirements for real world fault management. We present an overview of the architecture and the modeling language of SymCure. We review a sample of projects where we have applied this approach, and share the motivations, challenges, successes and failures that have been our companions along this memorable journey.

1 Introduction

Fault management plays a vital role across a broad spectrum of commercial and industrial applications, ranging from service level management and telecommunications network management in the Information Technology (IT) world, to abnormal condition management in manufacturing, chemical, oil and gas industries. The size and complexity of these applications often necessitates automated expert system support for fault management. A small number of root cause problems in IT communication networks often result in a large number of messages and alarms that cannot be handled by human operators in real time. Failure to identify and repair the root cause problems results in increased system downtime and poor service levels. Abnormal conditions in manufacturing and processing plants may result in unplanned shutdowns, equipment damage, safety hazards, reduced productivity, and poor quality products. A study funded by the US National Institute of Standards and Technology (NIST) estimates that in the absence of adequate fault management, billions of dollars are spent in addressing the problems caused by equipment failure, degradation, process drift, and operator overload [7].

There is increasing demand for the application of state-of-the-art fault management across a broad spectrum of industries. Over the past decade, we have applied model-based reasoning to address several fault management functions, including diagnosing root causes, testing them, predicting their impacts, and recovering from failures. This experience has guided us in the development of an object-oriented model-based domain-independent world fault management methodology, called SymCure (derived from "Symptom's Cure").

Section 2 describes our requirements for addressing large scale commercial fault management applications. Section 3 describes SymCure's architecture, reasoning algorithms, and modeling language. Our diagnostic methodology is largely domain independent and its applications range from abnormal condition management for offshore platforms that drill for oil at the bottom of the ocean to network management for satellite systems in space. Section 4 describes some of these applications. Section 5 concludes with a discussion of the lessons we have learned while applying our model-based diagnostic methodology in the real world.

2 Background

Fault management across various industries shares some common goals, such as improving application availability and utilization, reducing operator overload, and minimizing operation costs. In order to achieve these goals, it is necessary to develop fault management tools with the following capabilities.

Symptom monitoring. Symptoms are manifestations of underlying root causes and must be monitored to detect the occurrence of problems as soon as they happen.

Diagnosis identifies the root causes of known symptoms. (Diagnosis is also often referred to as fault isolation.) Complex systems are often composed of a large number of interconnected and interrelated components (e.g., networks of computers or satellites, electrical power generation plants, offshore oil drilling platforms). While a problem may originate on one component, often it is manifested on some other related component. In large-scale systems, it is not uncommon for multiple failures to overlap. Studies on such systems have shown that typically up to 80% of the fault management effort is spent in identifying root causes after the manifestation of symptoms [10].

Correlation. Modern systems are often richly instrumented with a large number of sensors that provide copious amounts of information in the form of messages and alarms. Often a small number of root causes result in a large number of messages and alarms that cannot be handled by human operators in real time. Therefore it is necessary to provide them with concise notifications of underlying root causes. Correlation is the process of recognizing and organizing groups of events that are related to each other. Usually such events share one or more root causes.

Prediction. Early prediction of the impacts of underlying root causes before the effects are manifested is critical for proactive maintenance, safety, and optimal system utilization. System operators need to know not just what impacts are predicted by the application but also when those impacts are expected to occur so they can plan for appropriate system repair and recovery.

Testing. In large systems, it is impractical and sometimes impossible to monitor every variable. Instead key observable variables are monitored to generate symptom events. Diagnostic inference typically identifies a set of suspected root causes. A test planning facility is needed to select additional variables to be examined to isolate the root causes. The fault management application then needs to request or run these tests, and utilize their results to complete the diagnosis. A test, as originally defined in [8], can incorporate arbitrarily complex analysis and actions, as long as it returns a true or false value.

Automated recovery. Identifying and automating recovery procedures facilitates rapid response to problems and allows for growth in equipment, processes, and services, without increasing the supervisory burden on system operators.

Notification. System operators require notifications of all critical fault management activity, especially the identification of root causes, and causal explanations for alarms, tests, and repair actions in a manner that they can follow easily. Sometimes they need to distinguish between what is observed by system sensors versus what is inferred by the underlying fault management application.

Postmortem. Information from diagnostic problem solving is fed back to the fault management system for historic record keeping and proactive fault management in the future.

24 hour, year-round fault management. The system topology may change at run-time over the life span of the fault management application, e.g., components may be added, removed, replaced, and modified. Commercial applications often require fault management on a 24 hour, year-round basis, so it may not be feasible to take the fault management system off-line each time that there is a change in the system topology.

A goal of ours for the past decade has been to develop a domain independent model-based fault management methodology to address these requirements.

With some effort at knowledge elicitation, it is often feasible to develop a high-level qualitative understanding of failure modes of system components and their effects on the behavior of the system. Such knowledge facilitates diagnostic reasoning based on qualitative fault models (e.g., [3], [5], [10], [11]). Alternative diagnosis techniques often described as consistency based methods (e.g., [1], [6]) use models of "normal" behavior, where diagnosis reasoning focuses on identifying causes for discrepancies between the normal behavior predicted by the model, and the aberrant behavior manifested by the device. Fault models can be far more abstract than models of normal behavior, and can be easier to construct, comprehend, and customize to a domain expert's specification. For example, fault models can easily capture causal relations such as "if an IP card fails, the device cannot communicate", and "if a pump fails, flow stops", without requiring detailed models that simulate normal behavior.

Causal fault models capture paths of causal interactions between root causes (i.e., faults) and their effects (i.e., symptoms). Diagnosing root causes from known symptoms is achieved by tracing upstream along the causal pathways from the symptoms to the faults. Predicting the impact of root causes is performed by propagating downstream from causes to effects.

In a number of commercial applications, such knowledge may be gleaned from domain experts, Failure Modes Effects and Analyses (FMEA) studies [9], and operational and product manuals. Domain experts need tools that allow them to model and analyze the structure and the diagnostic behaviors of their system. Object oriented graphical causal fault models facilitate understanding, capturing, updating, and reusing key elements of their diagnostic knowledge.

Our methodology, as detailed in the following section, utilizes graphical object-oriented qualitative causal fault models as the basis for several fault management functions, including diagnosis, correlation, prediction, testing, automated recovery, and notification.

3 SymCure Methodology

SymCure is implemented in G2 which is Gensym Corporation's graphical, object-oriented platform for developing and deploying expert systems applications.

Figure 1 shows the input, processing, and output elements of a SymCure application.

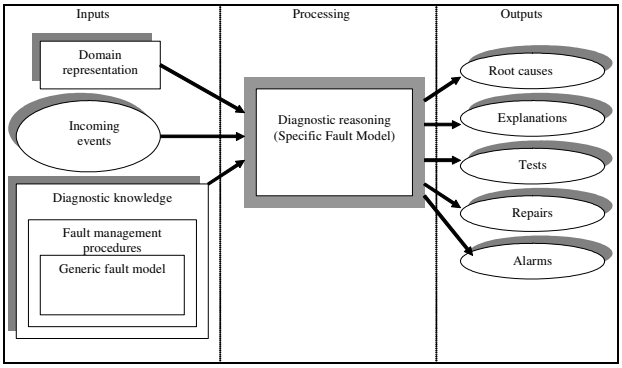


Figure 1. SymCure architecture

Domain representation is a graphical object oriented representation of the domain objects being managed by SymCure. It includes class definitions for domain objects, and a system description (also referred to as a *domain map*) comprised of specific instances (i.e., the managed domain objects) and relationships between these instances, including connectivity (e.g., one object is connected upstream of another) and containment (i.e., one object is contained inside another). Figure 2 shows two simplified illustrations of domain maps that are comprised of similar components (a furnace and a pair of pumps) but differ in their connectivity (i.e., Domain map 1 connects both pumps directly to the furnace, while Domain map 2 connects the pumps to the furnace in series).

Diagnostic knowledge comprises a declarative component, i.e., generic event-based fault models that are used to reach diagnostic conclusions, and a procedural component, i.e., test procedures that verify these conclusions, and recovery procedures that respond to these conclusions. We believe that separating out the declarative aspect of diagnostic knowledge (i.e., how do things fail, how do failures propagate) from the procedural aspect (i.e., what to do when something is suspected to have occurred, what to do when something is known to have failed) facilitates the articulation, acquisition, representation, and management of domain expertise.

A SymCure event[1] is a statement about a domain object that indicates the presence or absence of a problem. Generic fault models are composed of events (that are defined at the class-level) and their causal relations. Domain experts define these events using terminology they are familiar with, and at levels of abstraction suitable to their application. The fault models are graphical, thus they are easy to understand and little or no programming experience is necessary to build them. Another advantage of this approach is that the diagnostic knowledge does not require any reconfiguration whenever there are changes in specific instances of equipment, system topology (as illustrated by the two domain maps of Figure 2), or system operating modes, and they can be reused easily in different applications.

SymCure identifies a generic event as a unique combination of event name and target class (e.g., in the generic fault model for a pump in Figure 2, "Low flow" is the event's name and PUMP is its target class). Causal relations are represented by edges between generic events. Figure 2 shows simplified examples of generic fault models for pumps and furnaces. In the furnace generic fault model, low fluid flow into a furnace causes the temperature of the fluid to rise. At high temperatures, the fluid may undergo undesirable chemical reactions that, over time, cause carbon deposits inside the furnace tubes (i.e., fouling). Damage to a pump's impeller hinders its ability to impart motion to the fluid in the pump, thus causing low fluid flow through the pump. Inlet strainers are devices that keep debris out of a pump that might otherwise damage or clog it. Low flow through the pump is also caused by a plugged inlet strainer, which is also responsible for low inlet pressure. In these simplified models, low fluid flow into either the pump or furnace is caused by low flow in any domain object connected upstream of the pump or furnace. Propagation delays may be specified by configuring the properties of causal links; such delays are used to infer the occurrence time for any event. Although this is not shown in Figure 2, for illustrative purposes, we have assumed that there is a 3 hour delay between the onset of rising temperature in a furnace and the onset of fouling in the furnace.

The procedural component of a fault model includes the following elements.

1. Tests are used to verify the occurrence of an underlying event. Upon completion, a test action must return a true or false value for an associated event.
2. Repair actions are used to recover from failures.

Tests and repair actions are associated with underlying events. They are enabled or disabled by transitions of the values of their associated events. For example, in Figure 2, the test "Check inlet pressure" may be used to determine if the "Low inlet pressure" on a pump is true; and the repair action "Unclog inlet strainer" may be used to set in motion the process whereby a plugged inlet strainer is unclogged. In general, such actions may be automated, or may simply be a request to an operator, or a combination of the two, e.g., extracting a data point from a database or sending a repair technician to a remote site to conduct manual tests and repairs. A detailed discussion of tests and repair actions is beyond the scope of this paper.

[1] For the rest of this paper, we refer to a SymCure event simply as event.

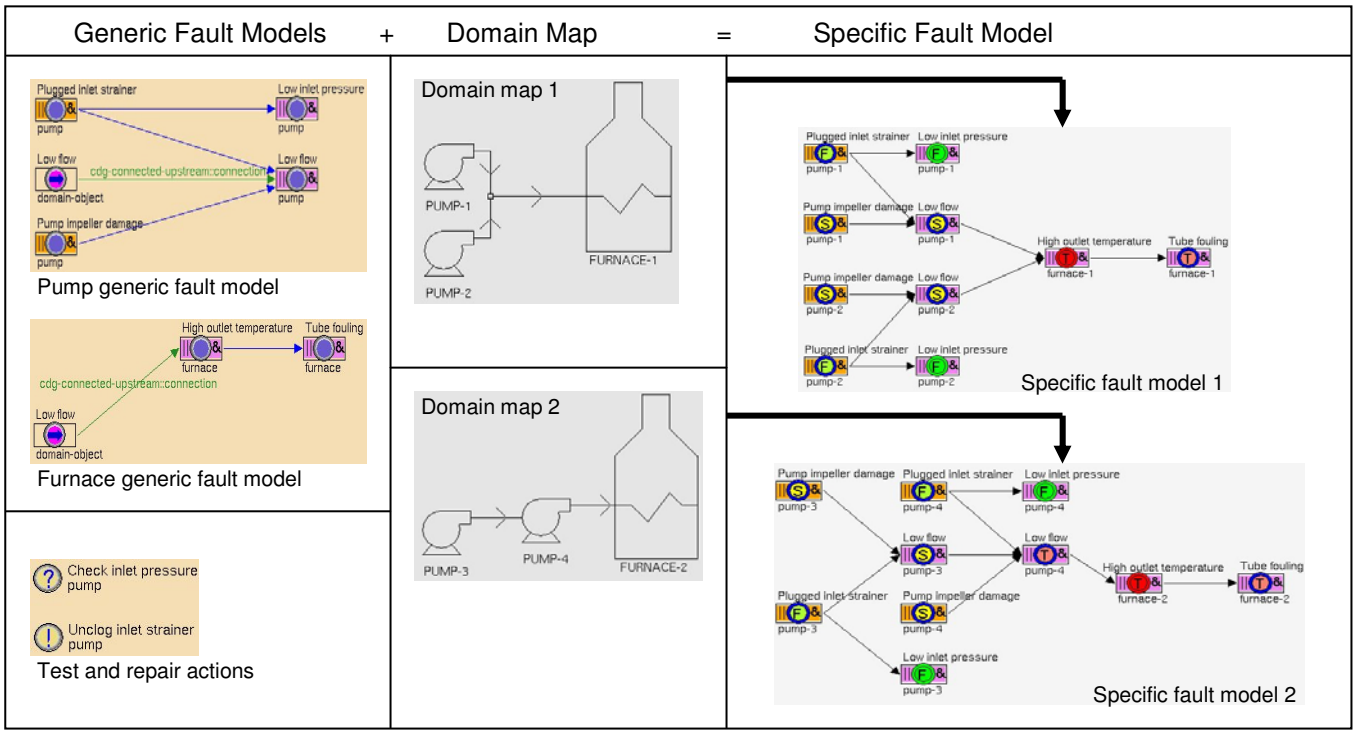


Figure 2. Generic fault models, domain maps, and specific fault models

The *incoming events* stream includes symptoms that indicate the presence of problems. Symptoms are manifestations of underlying root causes on specific domain objects. They are detected by application specific procedures[2] that monitor, aggregate, filter and analyze numerical data, sensor readings, network management traps, signals, and other forms of raw data. SymCure responds to an incoming event by diagnosing the root causes of the incoming event stream, predicting the impact of the root causes on other events, reporting the results of diagnosis and impact prediction to system operators, initiating any tests required to verify the occurrence of suspected root causes, and initiating recovery actions to repair the target objects of the root causes. Incoming events also include test results that validate or rule out suspected events.

SymCure's diagnostic algorithms dynamically combine the domain representation, diagnostic knowledge, and incoming events to produce a *specific fault model* that applies to the specific managed domain objects. A specific fault model is composed of specific events and their causal relations. In Figure 2, specific fault models 1 and 2 are derived from domain maps 1 and 2, respectively, and the generic fault models for pumps and furnaces. SymCure uniquely identifies a specific event by its name (e.g., "Low flow") and the domain object on which the event occurs (e.g., PUMP-1). For diagnostic reasoning purposes, specific events may take on the following symbolic values:

- true, i.e., the event is known or inferred to have occurred; such an event is depicted with "T" in the specific fault model (e.g., "High outlet temperature" on FURNACE-1 in Figure 2),
- false, i.e., the event is known or inferred to not have occurred; such an event is depicted with "F" in the specific fault model (e.g., "High outlet pressure" on PUMP-1 in Figure 2),
- unknown, i.e., it is neither known nor inferred that the event has occurred; such an event is depicted with "U" in the specific fault model, and
- suspect, i.e., it is suspected that the event has occurred; such an event is depicted with "S" in the specific fault model (e.g., "Pump Impeller damage" on PUMP-1 in Figure 2).

The specific fault models in Figure 2 are initiated by assuming that high outlet temperature is observed in each furnace and manual testing shows the inlet pressures on all pumps are normal. For each specific fault model, SymCure infers that the impellers on one or both pumps must be damaged. For each scenario, SymCure also predicts that tube fouling will occur.

We use best first search to propagate event values within a specific fault model. At a very high level (for details refer to [3]), starting from an incoming event, event propagation is achieved by the following algorithm.
for any event *e* if its value changes **do**

[2] Event detection is outside the scope of this paper; refer to [3] for further details.

1. propagate the value of the event upstream to all *Causes* (where *Causes* = all causes of *e*);
2. propagate the value of the event downstream to *Effects* (where *Effects* = all effects of {*Causes* + *e*});

end for

This for-loop is invoked each time a symptom is manifested and the result of a test becomes available. The propagation steps in the for-loop can be configured to terminate as soon as SymCure identifies a set of root causes that explain all observed incoming events, even before a complete specific fault model has been constructed. To speed up the search process, in steps 1 and 2 SymCure explores events on the same domain object before it moves on to events defined on different domain objects. The time complexity of the for-loop is linear in the number of events (i.e., size) of the specific fault model. The maximum number of events in a specific fault model is bound by the product of the number of managed domain objects and the size of the largest generic fault model. In practice, since we construct only the events that are correlated to incoming symptoms, the actual size of a specific fault model is usually a small subset of the maximum possible size.

As demonstrated in Figure 2, SymCure constructs system wide specific fault models, i.e., it is able to reason about interactions among the individual system components. We generate a visual representation of the underlying specific fault model on demand. This allows modelers and system operators to understand the results of the diagnostic reasoning process. However, graphical models can rapidly overwhelm users as the number of nodes increase. Thus, we provide options to organize the visual representation of a specific fault model by focusing on relevant portions of the graph at any given time. For example, the abridged specific fault model in Figure 3 shows only the potential causes and effects of "High outlet temperature" in FURNACE-1.

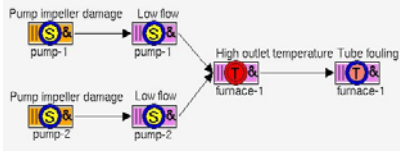


Figure 3. A focused view of Specific fault model 1

SymCure generates messages for root causes and alarms, and proposes tests and repair actions to resolve and recover from faults. Such messages are presented to system operators in one or more diagnostic console browsers. Figure 4 shows the alarms, root causes, tests, and repair actions for Specific fault model 1 in Figure 2. Note that tube fouling is predicted to occur 3 hours after high outlet temperature in the furnace has become true. Alarm messages can be suppressed in favor of root causes so that operators are not flooded with alarms. Requests for tests and repair actions may be sent automatically to a rudimentary workflow management component, which schedules and executes them.

Severity	Message
ALARM	Tube fouling on furnace-1 will become true (downstream inferred) at 2/23/2007 13:14:18
ALARM	High outlet temperature on furnace-1 has become true (specified) at 2/23/2007 10:14:18
ALARM	Low flow on pump-1 is suspect (upstream inferred)
ALARM	Low flow on pump-2 is suspect (upstream inferred)
ALARM	Low inlet pressure on pump-1 is false (specified)
ALARM	Low inlet pressure on pump-2 is false (specified)
REPAIR-ACTION	Unclog inlet strainer on pump-1 is CREATED
REPAIR-ACTION	Unclog inlet strainer on pump-2 is CREATED
ROOT-CAUSE	Pump impeller damage on pump-1 is suspect (upstream inferred)
ROOT-CAUSE	Pump impeller damage on pump-2 is suspect (upstream inferred)
ROOT-CAUSE	Plugged inlet strainer on pump-1 is false (upstream inferred)
ROOT-CAUSE	Plugged inlet strainer on pump-2 is false (upstream inferred)
TEST	Check inlet pressure on pump-1 is INACTIVE
TEST	Check inlet pressure on pump-2 is INACTIVE

Figure 4. Alarms, repair actions, root causes, and tests for Specific fault model 1

SymCure's modeling language. Early versions of SymCure provided two kinds of events for creating fault models: OR event and AND event. Their behaviors were governed by the following rules (where X_i, Y_j, and Z_k are events in a fault model; the edge $X_i \rightarrow Y_j$ implies that X_i causes Y_j; and propagating downstream requires determining a value for Y_j from X_i, while propagating upstream requires determining X_i from Y_j).

For any OR event Y_i

1. While propagating downstream, if $Y_i \rightarrow Z_1,\ldots,Z_n$, and Y_i is true, then $Z_1,\ldots,Z_n$ are all true.
2. While propagating upstream, if $X_1,\ldots,X_m \rightarrow Y_j$, and Y_i is true, then at least one of $X_1,\ldots,X_m$ must be true.

For any AND event Y_i:

1. While propagating downstream (identical to the case of an OR event), if $Y_i \rightarrow Z_1,\ldots,Z_n$, and Y_i is true, then $Z_1,\ldots,Z_n$ are all true.
2. While propagating upstream, if $X_1,\ldots,X_m \rightarrow Y_i$, and Y_i is true, then all of $X_1,\ldots,X_m$ must be true.

The limitations of a causal modeling language comprised solely of these two events were quickly exposed in several early projects. In the real world, fault models are rarely ever complete, i.e., frequently there are causal influences that are not fully understood or cannot be modeled. Propagation delays and noise often result in discrepancies between inferences made by the underlying fault model and observed events. This may result in root cause events being exonerated prematurely or being implicated falsely. We have tried to address these issues in later versions of SymCure by associating OR and AND logic at the input and output of an event as shown below:

1. While propagating downstream, if $Y_i \rightarrow Z_1,\ldots,Z_n$, and Y_i is true, then $Z_1,\ldots,Z_n$ are all true (we say that Y_i uses output AND logic).
2. While propagating downstream, if $Y_i \rightarrow Z_1,\ldots,Z_n$, and Y_i is true, then at least one of $Z_1,\ldots,Z_n$ must be true (we say that Y_i uses output OR logic).
3. While propagating upstream, if $X_1,\ldots,X_m \rightarrow Y_i$, and Y_i is true, then at least one of $X_1,\ldots,X_m$ must be true (we say that Y_i uses input OR logic).

4. While propagating upstream, if $X_1,\ldots,X_m \rightarrow Y_i$, and Y_i is true, then all of $X_1,\ldots,X_m$ must be true (we say that Y_i uses input AND logic).

We can also specify input and output logic with percentages to reason over a progression of values that may represent partial degradation. By combining these rules, we have created 7 different events are capable of root cause analyses and impact predictions that cannot be modeled with traditional OR and AND events alone. Further modeling enhancements, including utilizing NOT logic, mutual exclusion, and specifying state dependent behavior. See [3] for detailed examples of SymCure's capabilities and modeling language.

4 Applications

SymCure's diagnostic knowledge representation and reasoning methodology is domain independent and it has been applied to solve problems in domains ranging from networks of telecommunication satellites to offshore oil drilling platforms. We present a sample of such applications emphasizing, wherever applicable, the challenges they posed to our methodology and its consequent evolution.

Satellite network fault management (1995-2000). Iridium is a global telephony network of low earth orbit satellites and ground stations where calls are switched in the sky from satellite to satellite; as the satellites orbit the earth, communication links are periodically broken and are re-formed with different satellites. Diagnosing the root causes of communication failures required reasoning over dynamically changing communication links, so Motorola (Iridium's original owner) adopted an early version of SymCure for its satellite network fault management application. Experience with live monitoring before satellite launches, prior to full-scale deployment suggests that, had the fault management application been fully deployed, it might have saved some satellites that were lost just after launch [11]. To the best of our knowledge, the application was in use until 2000. Later, beset with financial difficulties and other business problems, Iridium ran out of funds to support the application.

Enterprise-wide monitoring of networks and applications (1998-2000). BMC built a line of products (aimed at Windows 2000 and Microsoft Exchange servers) for diagnosing faults and predicting their impacts on enterprise-wide computer networks and applications. The requirement to automatically diagnose and predict events over dynamically changing network topology and software installed on servers, led them to use SymCure as their fault management reasoning engine. This work tested some of the limitations of earlier versions of SymCure's modeling language because of timing delays along paths of causal propagation (primarily due to the lag times between measurements and their time-averaged thresholds), noise, and incomplete causal models.

Highway traffic management equipment monitoring (2001-02). L.E.E. developed a SymCure application to monitor and manage Paris' highway traffic equipment including thousands of road sensors, message displays, cameras, telecommunications equipment, hierarchical subsystems, energy systems, and supporting network infrastructure. Diagnostic conclusions are required within a short (2 minute) event polling cycle, of which only a fraction (approximately 20 seconds) is available for diagnostic processing. Early versions of SymCure built visual representations of complete specific fault models and reasoned over them. Drawing and updating events at run-time is inefficient and contributed to the difficulties in meeting the performance requirements. We realized that sound software design is as vital as "intelligent" technology for a successful application. In subsequent versions of SymCure, visual representations are separated from underlying specific fault models; they are generated on demand only and usually focus on a narrow section of a specific fault model as illustrated by Figure 3 in the previous section. Along with the introduction of best first search and efficient data structures like hash tables instead of lists, this has resulted in a five-fold improvement in the speed of diagnostic reasoning.

Heaters in oil refineries (2001-02). A middle eastern oil refinery installed an application for managing abnormal conditions, focused on diagnosing failures in a generic class of heaters. The application defined 80 root causes that account for over 240 different kinds of operator messages, and provided tests and repair actions that rapidly guide operators to return a heater to normal operation. Noureldin and Roveta [4] describe this application in detail including how they acquire domain knowledge from human experts. They concluded that the net savings from using this application substantially exceeded the cost of the project in less than one year.

Offshore oil production platform (2002-03). Halliburton KBR created an application for monitoring the health of offshore oil production platforms. Their application focused on monitoring the health of the gas compressors on a platform and demonstrated its ability to proactively predict compressor failures that could potentially save millions of dollars that would otherwise be sacrificed to production losses resulting from compressor shutdowns [2].

Mineral processing plant (2005-present). SGS MinnovEX is developing an application to diagnose and respond to root causes of sub-optimal operations in mineral processing plants. System components include equipment such as compressors and pumps, controllers, and other sub-processes. The application analyzes

historical operational data to detect deviant operational events and then diagnoses the causes of such events. This application uses SymCure's enhanced event logic and preliminary results indicate success in identifying faults such as sensor drift and poor controller tuning in the face of incomplete models and noisy data. However enhancing the event logic complicates the behaviors of the fault models. This motivated us to build a graphical fault model debugger to help domain experts to step through, analyze, and test their fault models.

Electrical power and energy distribution systems (2005-present). General Atomics is developing a generic fault management library for mission critical power and energy distribution systems. Though a significant degree of redundancy is typically designed into critical power systems, the penalties associated with loss of availability are so high that any software assistance in the detection of the onset of failure becomes invaluable. Generic fault models have been developed for electrical system components including generators, motors, circuit breakers, transformers, uninterruptible power supplies, transfer switches, critical load centers, and DC links. These models enable the diagnosis of complex power system anomalies such as ground faults, which typically cause a flood of alarms. The strengths of generic fault modeling, in conjunction with dynamic instantiation of energy flow relationships between electrical components, have effectively diagnosed problems in dynamically reconfigurable power systems. Combining SymCure's fault management methodology with models of normal behavior that analyze trends and detect discrepancies between observed and predicted sensor values, facilitates the prediction of the onset of equipment failure. SymCure's graphical fault models facilitate conveying, communicating, and validating the details of vendor-supplied Failure Modes and Effects Analysis (FMEA).

General Atomics' fault management library serves as the basis for several ongoing projects with the US Navy, which is increasing its reliance on high-energy electromagnetic components for next generation shipboard systems. This includes projects to build electromagnetic systems to impart the momentum necessary to launch airplanes from an aircraft carrier and that arrest the motion of the planes to bring them to a halt when they land. (The runway in an aircraft carrier is too small for airplanes to be able to take off and land without such assistance.) As availability and reliability are critical for such systems, General Atomics is leveraging SymCure's fault management methodology to diagnose and predict the health and life expectancy of aircraft launch and landing gear components.

General Atomics is also collaborating with the National Aeronautics and Space Administration (NASA) for fault management of its Rocket Engine Test Stand (RETS). RETS tests the structural integrity of a rocket before it is launched, and typically provides NASA's engineers their last chance to detect and correct any flaws in the fully assembled rocket. The fault management application is targeted for deployment at NASA's Stennis Space Center facility in southern Mississippi. The subsystems associated with the testing of solid and liquid fueled rocket engines involve complex mechanical, electrical, hydraulic, pneumatic, and thermodynamic processes. All of these processes, along with their associated system components, lend themselves well to the generic fault modeling capabilities of SymCure. The fault management application will be used to diagnose and predict RETS anomalies, but will also provide real-time advisories associated with the quality of results obtained from the RETS tests.

5 Conclusions

In this paper, we have described a generic model-based fault management methodology to address fault management applications in several diverse domains. We summarize our contributions to the application of model-based diagnostic techniques in the real world, limitations of our methodology, and share our thoughts on the lessons we have learned.

Contributions. SymCure integrates causal fault models for root cause analysis and impact predictions with test and repair management thus providing a comprehensive set of capabilities for developing fault management applications. SymCure's modeling language has evolved over time to address problems with propagation delays, sensor noise and threshold problems. SymCure's reasoning process is capable of detecting and resolving multiple system failures. SymCure's fault models are generic, so they do not require any reconfiguration whenever there are changes in equipment, system topology, or system operating modes and they can be reused in different applications.

Limitations. Like any knowledge based reasoning system, the accuracy of SymCure's diagnostic inference is constrained by the correctness of the underlying fault models and the availability of instrumentation to observe symptoms and perform tests to resolve diagnostic candidates. SymCure cannot handle faults that are not part of the fault models but it can identify novel combinations of known faults. Because SymCure processes an event as soon as it is received, diagnostic results are susceptible to the order of incoming events. In theory, this can cause problems over short time spans if the values of observed events are inconsistent (because of propagation delays, noise, and faulty sensors). In practice, this has not been a significant issue. SymCure allows domain experts to represent events in their terminology at an arbitrary level of abstraction suitable to their application. However, the burden of detecting the

occurrence of an event is placed on external monitoring mechanisms, which may require sophisticated filtering and aggregation techniques. SymCure does not explicitly model the likelihoods (i.e., probabilities) of events.

Lessons learned. There are tremendous and exciting opportunities for the application of state-of-the-art fault management techniques in commercial settings and, with billions of dollars at stake, industries are eager to embrace intelligent knowledge based solutions. However, as we have learned over the years, there are many challenges and pitfalls of which practitioners need to be aware.

The fundamental challenge for knowledge based expert systems applications is the formalization of the domain knowledge both before and during the development of the application. Often the knowledge base evolves incrementally, starting from a small prototype that grows with further testing and development as new requirements become evident. Commercial applications require ease of use so they can be mastered with minimal training and cost. This motivates not just the development of powerful user interfaces, but also modeling languages that can be understood by users whose expertise is limited to their domain. Modeling aids to efficiently search for, navigate to, and debug diagnostic knowledge are essential. If such an application is to be adopted successfully in a commercial setting, it must be able to demonstrate a reasonable return on investment. There is a premium on "out of the box" solutions, where the diagnostic knowledge is pre-packaged. However, it is imperative that this knowledge be configurable by end users, since we have never seen two domain experts ever agree on everything. Visual representations and explanations are a crucial ingredient for success; a picture is worth not just a thousand words but potentially millions of dollars, if it helps system operators to avert an impending problem. Efficient computational techniques and sound software engineering principles are as important as the artificial intelligence that underlies its reasoning capabilities. Last, but not the least, human factors, including sky high expectations, funding cuts, resistance to adopt new technology, political infighting, and employee turnover (especially, if it involves someone who is a champion of the technology) are, not infrequently, serious impediments to success.

As described in the previous section, we have plumbed the depths of oceans (with offshore oil production platforms) and orbited the heavens (with telecommunication satellites). We believe that the road ahead promises to be just as exciting. As the future unfolds, we are embarking on the development of generic fault model libraries of components in different domains that can be reused across applications and other innovative techniques to automate knowledge discovery and to create adaptive fault models that learn from successful and failed diagnoses.

References

1. G. Biswas, R. Kapadia, and X. Yu. "Combined Qualitative-Quantitative Diagnosis of Continuous valued Systems". In IEEE Sys. Man, and Cybernetics. Vol 27, No. 2, pp. 167-185, March 1997.
2. R. Guddeti. "An Expert System for Real-Time Monitoring of Offshore Platform Equipment". GUS 2003 Gensym User Society Worldwide Meeting, Boston, Massachusetts, April 2003.
3. R. Kapadia. "SymCure: A model-based approach for fault management with causal directed graphs", Proc. of the 16th Intl. Conf. IEA/AIE-03, pp. 582-591, Loughborough, UK. June 2003.
4. H. Noureldin and F. Roveta. "Using Expert System and Object Technology for Abnormal Condition Management", BIAS 2002 International Conference, Milano, Italy. November 2002.
5. M. Porcheron and B. Ricard. "Model-based diagnosis for reactor coolant pumps of EDF nuclear power plants". Proc. of the 10th Intl. Conf. IEA/AIE 97, pp. 411-420, Atlanta, USA, June 1997.
6. Readings in Model-based Diagnosis. W. Hamscher, L. Console, J. deKleer, eds. Morgan Kaufman, 1992.
7. D. Siegel. "Abnormal Condition Management: Minimizing Process Disruptions and Sustaining Performance through Expert Systems Technology". Natl. Petroleum Refiners Assn. 2001 Comp. Conf., Dallas, Texas, October 2001.
8. W. Simpson and J. Sheppard. "System Test and Diagnosis". Kluwer Academic Publishers, Boston 1994.
9. D. Stamatis. "Failure Modes and Effect Analysis", ASQ Quality Press. 2003.
10. G. Stanley and R. Vaidhyanathan. "A Generic Fault Propagation Modeling Approach to On-line Diagnosis and Event Correlation". Proc. of the 3rd IFAC Workshop on On-line Fault Detection and Supervision in the Chemical Process Industries, Solaize, France, June 1998.
11. R. Stewart, G. Stanley, and V. Vasudevan. "Integrated Fault Management in a Satellite-Based Global Telecommunications Network", Software Engineering Symposium 1995, Ft. Lauderdale, Florida, June 1995.

Gradient-based Diagnosis

Willibald Krenn, Franz Wotawa*

Abstract

Repairing a fault in a highly dynamic environment under uncertain conditions not always requires a distinct fault detection and localization pre-processing step. In this paper we present an approach based on the behavior of a device combined with gradients which indicate the appropriateness of a certain behavior in a given situation. The gradients in our approach work like a memory storing past experiences and thus allow for adapting the system's behavior smoothly over time. We show that the approach is feasible for small devices having strong resources constraints regarding computational power and available memory. Moreover, we introduce a case study where a device using the gradient approach is capable to handle permanent and transient faults which occur over time.

Introduction

Devices designed for autonomous and mobile use are optimized for low energy consumption. A side effect of this optimization often times is low processing power, little onboard memory, and only few redundant hardware modules. This is especially true in existing designs not geared toward intelligent control. While these conditions are quite limiting, there is a subclass of autonomous and mobile devices that have to provide only simple functionality (e.g. a remote sensor) and, thus, do not require huge modeling effort. That said, we faced the challenge to add adaptive behavior and diagnosis capabilities to such a system in order to extend the durability of said device.

Implementing adaptive behavior and diagnosis for this kind of mobile devices means implementing planning, replanning, diagnosis, and repair capabilities within one framework that has to be as cut down as possible. Due to this constraint and other requirements that called for a reactive design, we pursued an integrated approach to solve the over all control problem. Dedicated algorithms for each of these subtasks would have lead to an overstretch of resource consumption, hampering the reactiveness of the device. The integrated approach emphasizes the control and repair aspect of the over all control problem, optimizing CPU and memory costs by omitting explicit fault localization.

*Technische Universität Graz, Institute for Software Technology, Inffeldgasse 16b/2, A-8010 Graz, Austria, {wkrenn,wotawa}@ist.tugraz.at

Our system model comprises a knowledge base and a set of goals the system has to attain. In particular we request the knowledge base to contain the complete knowledge about how a certain goal can be attained. We do *not* request the knowledge base to contain explicit information about what to do in case of component failures or other environmental conditions preventing a certain goal can be attained. In addition, we do *not* require the knowledge base to contain information about failure states or some special system model only used for diagnosis purposes. We do not even care to differentiate between permanent and transient faults: In our view, a permanent fault is a special kind of transient fault. Strictly speaking, we do not require a model of the external environment to be included in the knowledge base. While some limited model can be included, we argue that any model of the environment is incomplete and will not model all effects that can cause transient and/or permanent faults appear and disappear. Consequently, we are physically unable to decide whether a fault is permanent or transient. While we do not differentiate between the two classes of faults, we distinguish between the amount of time a fault is present. All these properties help us minimizing the size of the knowledge base, which in turn is useful as storage space is a very limited resource.

This paper is organized as follows: At first, we present some basic definitions. Thereafter we show how control and diagnosis in terms of repair are working hand in hand in order to assess each rule. This is done in a section called "Calculating Weights", which is completed by a comprehensive example. The subsequent sections present an overview of an algorithm implementing this control system, discuss how diagnosis in terms of fault localization relates to the presented approach, and give results gained from a case study. The paper closes with a discussion of related research and a conclusion summing up key aspects of the presented methodology while also mentioning areas left for future research.

Basic Definitions

As already mentioned our model combines classical rule-based elements with gradients that are used to model effects found in biological systems. We use rules for representing possible behaviors that are required for reaching pre-defined goals. Behaviors are actions and tests regarding the current state of the world. In contrast to rules the gradients are de-

fined for rule consequents and specify the degree of priority for reaching the consequent. During the computation the gradients are adapted in accordance to the previous experiences in reaching the consequent. For example, if a consequent can never be reached because of a broken system component, the corresponding gradient value is sub-sequentially reduced until it reaches it minimum value. Hence, alternatives for reaching a similar consequent but using different tests and actions are preferred.

As example, imagine a remote sensor platform that has two redundant communication channels on its disposal. Both channels are used to transmit sensor data but under normal operating conditions, one channel is strongly preferred over the other one. In our model, the previous requirement will lead to two different gradients attached to the consequents of the rules modeling data transmission: Simply speaking, the preferred channel will have a gradient attached that signals higher importance than the one attached to the backup channel. If we continue with our thought experiment and simulate a fault on the preferred communication channel, we will see that nothing changes at first. Over time, however, the gradient of the preferred – and now blocked – communication channel will get smaller until the importance of the two channels is inverted. The change in importance is reflected in the ratio of the usage of both channels. As soon as the backup channel is more important, it will be used more often than the blocked primary channel. Every now and then, however, the system will use the primary channel and "test" whether the channel is working again. In case the transient failure is gone, the system will smoothly revert back to the "normal" operational behavior over time. In the remainder of this paper, we will come back to this example and formalize it according to our theory.

Definition 1 (System Knowledge Base (SKB)). *The system knowledge base (SKB) is a tuple* (P, R, G, γ) *where*

- P *is a set of propositions.*
- R *is a set of definite horn-clauses of the form* $x_1 \wedge \ldots \wedge x_n \rightarrow y$ *where* $x_1, \ldots, x_n, y \in P$. $x_1 \wedge \ldots \wedge x_n$ *is the antecedence and* y *is the consequent.*
- $G \subseteq P$ *is a set of goals. Goals are marked consequents of* R.
- $\gamma : P \times \mathcal{R} \mapsto \mathcal{R}$ *is a function mapping a real number to a real number for one consequent in* P.

Note that propositions can represent actions, i.e., activities which might alter the state of the system or the environment, e.g., `TurnOnGps` which turns on the GPS, basic propositions which are used to store certain knowledge, e.g., the current position of the system which has been measured by the GPS, and goals. In order to distinguish the different kind of propositions we assume that actions are represented by `action(`X`)`, and goals are of the form X`:`.

The purpose of our intelligent control system now is to determine the execution of certain rules specified in SKB in order to reach the goals. Hence, we are searching for a sequence of rules $\langle r_1, \ldots, r_k \rangle$ where $\forall\, i \in \{1, \ldots, k\} : r_i \in R$ so that the sequential application of rules lead to the goals when starting with the current state S. More formally, we define states and SKB executions as follows:

Definition 2 (State). *A (system) state* $S \subseteq P$ *is a set of propositions that are known to be true. Propositions that are not in a state are assumed to be false in that state.*

Definition 3 (Execution). *Given a SKB. An execution* e *of SKB from state* S *is a sequence of rules* $\langle r_1, \ldots, r_k \rangle$ *where* $\forall\, i \in \{1, \ldots, k\} : r_i \in R$ *so that* $S \wedge r_1 \models S_1$, $S_1 \wedge r_2 \models S_2$, …, $S_{k-1} \wedge r_k \models S'$ *with* $S' \subseteq G$ *under the assumption that all actions will succeed. We write* $S \cup e \models S'$.

Definition 4 (Length And Cardinality). *Given an execution* $e = \langle r_1, \ldots, r_k \rangle \wedge S \cup e \models S'$. *The length of* e *is* k. *The cardinality of* e *is the cardinality of* S'.

Note that we do not require an execution to fulfill all goals because this might not be possible in a certain situation because of failing actions or environmental conditions like a missing wireless network when adapting to send information. Moreover, the definition does not restrict an execution to be comprised of some minimal number of rules. It also does not restrict an execution to use a rule only once.

As our rule based system does not allow the revocation (negation) of a calculated fact (proposition), we are able to deduce all facts by including each rule once. Therefore, we define a minimal execution that disallows a rule to be included more than once and even goes beyond that by ruling out alternatives within the execution.

Definition 5 (Minimal Execution). *Given an Execution* e. *The execution is minimal iff* $\nexists l, k : 0 < l \neq k <= length(e) \wedge Consequent(r_k) = Consequent(r_l)$ *with* $r_k, r_l \in e$.

For a given SKB and a certain state, there may exist several different minimal executions. Because of a different number of entailed goals, component failures, environmental conditions, or design requirements, not all minimal executions are equally well suited. The minimal execution best suited is called preferred execution.

Definition 6 (Goal Optimality). *Given a SKB and its set of minimal executions* $E = \langle e_1, \ldots, e_n \rangle$. *A minimal execution is said to be goal optimal, iff its cardinality is greater or equal to the cardinality of all other minimal executions in the set.*

The goal optimality and minimal execution definitions ensure that the given knowledge is used as much as possible to entail a maximum number of goals while focusing on smaller rule sequences. However, there might be some goals which are less important than others. Moreover, in practical applications not all actions can be executed because of faults. This information has to be stored and used over time for selecting the best execution. For this purpose our SKB comprises a weight function for rule consequents which is dynamically adapted during rule execution. For example, if the application of an action, e.g., turning on the GPS fails, the weight of the consequent is reduced. If the action passes again, the weight is increased. In this way the system stores information about past executions. This weight adaption is done using a gradient function. We discuss this issue later. Let us first define optimality with regard to the weight.

Definition 7 (Preferred Execution). *Given a set of minimal, goal optimal executions* M *starting in a common state. A*

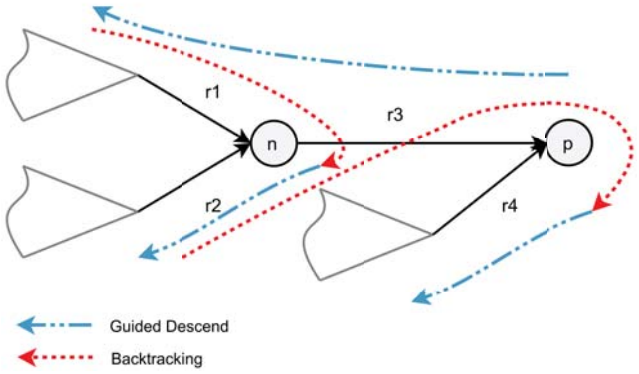


Figure 1: Searching the preferred path: Rules r1 to r4 have "n" or "p" as rule-consequents, where "p" also is a goal. The system tries to find a sequence of rules that make "p" true.

minimal, goal optimal execution $e \in M$ is a preferred execution iff for all consequents $c \in e : Weight(c, e) >= Weight(c, a_i)$ where $i = 1 \ldots |M| \wedge a_i \in M$.

Note that it is likely that a consequent c is not part of another minimal execution $a_i \in M$. In this case, $Weight(c, e) >= Weight(c, a_i)$ is always fulfilled. We defer the discussion of how the weight is being calculated to the next section.

Example 1. The intention behind the weight comparison is to introduce a guided local-best search. Figure 1 shows four rules (r1, r2, r3, r4) that span three different minimal executions: e1:$\langle r3, r1\rangle$, e2:$\langle r3, r2\rangle$, and e3:$\langle r4\rangle$.

We assume the weight of consequent n in rule r1 to be greater than the weight of consequent n in rule r2 and the weight of consequent p to be greater in rule r3 than in rule r4. When doing the top-down search, this weight assignment leads to e1 or e2 being candidates for preferred execution in the first step. In the next and final step, we finish the calculation with e1 being the preferred execution.

So far we have presented a theory that deals with one execution only. In practice, however, the system needs to have an infinite sequence of executions. Unfortunately, we often cannot use a state $S_1 : S_0 \cup e_1 \models S_1$ as a starting point for another execution $e_2 : S_1 \cup e_2 \models S_2$. (With $S_0, S_1, S_2 \subseteq P \wedge e_1, e_2$ preferred executions .) The reason for this inability to use the end-state of one execution as start state for another one are the facts that have been deduced during the last execution. In case an execution attained goals, the system therefore needs to forget all the facts that have led to the goals. More formally, we define a run of an SKB as follows.

Definition 8 (Run)**.** *A sequence of preferred executions $e_0 \ldots e_{n-1}$ together with functions α and β form a run $[\beta(S_0, \alpha) \wedge e_0 \models S_1], \ldots, [\beta(S_{n-1}, \alpha) \wedge e_{n-1} \models S_n]$ where $\alpha : G \mapsto P$ calculates all consequents that were necessary to reach a goal and $\beta : P \times X \mapsto P$ removes a set $X \subseteq P$ from the current state and updates the gradients.*

In practice, of course, the run will last until the device does a shut down, or suffers severe damage. After presenting the main theory of our approach, the next section will cover weight calculation.

Calculating Weights

Knowledge about the current system state, which implies knowledge about faults, is stored in the SKB in two ways: On the one hand there is predefined knowledge, namely propositions, rules, and initial activity profiles, on the other hand, the system also stores knowledge gained during a run in the SKB. (This extends the definition of SKB.) Latter kind of knowledge is present in the form of a damping - and an activity factor. During the run, as a goal has been attained or failed, the SKB gets updated accordingly. Figure 2 shows how the weight of a consequent is extracted from γ. (The function γ encodes the preferred activity profile of each consequent.) In Figure 2 "D" stands for a damping factor that is dynamically adjusted during the run, "L" stands for the distance to a local maximum of γ, and "S" is the value of γ' at the current activity value. Thereby γ' is the first derivative of γ: $\gamma' = \frac{\partial\gamma(P,activity)}{\partial activity}$. Weight is calculated by some function $\delta : \Gamma \times \mathcal{R} \times \mathcal{R} \mapsto \mathcal{R}$ that – in a simple case – returns a value obtained by $\gamma'(P, activity) * L * (1 - D)$. Note that Γ is the set of all possible functions $P \times \mathcal{R} \mapsto \mathcal{R}$ where $\mathcal{R}$ denotes the real numbers and P the consequents (propositions) of the rules.

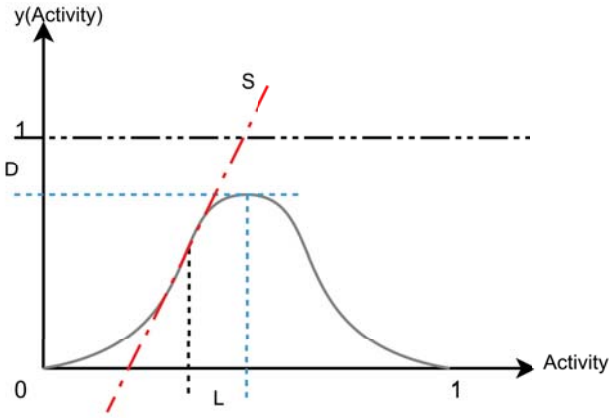


Figure 2: Using function γ and an activity counter to calculate the weight for a consequent of a rule: $Weight = S * L * (1 - D)$

While this methodology does not provide means for exact fault localization, damping factors together with the rule set allow an approximative localization of faults, which is sufficient for our intended application. One important property of weight calculation in connection with the algorithm presented in the next section is that the system tries to minimize all weights and, thus, get the activity factor of each consequent to the value specified by γ. This reflects the idea that the complete system behavior – as seen from outside – is a superposition of several embedded, smaller behaviors. In our approach, the atoms of "small behavior" are all plans stored inside the knowledge base telling the system how to

attain a certain goal. The superposition of behaviors not only depends on whether plans meet their preconditions and are therefore run but also on γ that tells the system how often, e.g., a goal shall be reached in the first place. The latter property is important for being able to control the system behavior, biasing it according to the situation the device is in. Our approach features two distinct mechanisms that can be used for strategy control: The first mechanism is using the damping factors. The second one can be exploited by user-supplied code, e.g., inside actions and permits a change of γ at any time.

Example 2. We extend the first example and show how weight calculation works under the assumption that δ computes the weight by $\gamma'_n(P, activity) * L * (1 - D)$ where γ'_n calculates a normalized slope.

Following rules are used in this example:
(r1) $n \leftarrow x \wedge action(\texttt{SendPosGSM})$
(r2) $n \leftarrow x \wedge action(\texttt{SendPosNF})$
(r3) $p \leftarrow n \wedge action(\texttt{StoreSendTime})$
(r4) $p \leftarrow x$
For the sake of simplicity, we assume that proposition "x" is always true. Rule four does not do anything meaningful in this example and is mainly used to demonstrate that all alternatives are selected by the system over time. The intention of the knowledge engineer is to slightly prefer sending the position by running the action `SendPosGSM`. Furthermore, the knowledge engineer has determined γ for each rule, the decay value for the system, and increment values for damping and activity factors:
$\gamma_{max} = 1$
$\gamma(P, 0)'_n = 0.5$
$\gamma(P, 0.25)'_n = 0.25$
$\gamma(P, 0.5)'_n = 0.5$
$\gamma(P, 0.75)'_n = 0.3$
$\gamma(P, 0.875)'_n = 0.2$
$activity_{incr} = 1$, $damping_{incr} = 0.2$
$D_{r1} = 0$, $D_{r2} = 0.2$, $D_{r3} = 0$, $D_{r4} = 0.9$

We now calculate the weights of all minimal executions for a run of a sequence of four preferred executions.

1. $\left.\begin{array}{l}\delta_{r1}(0.5, 1.0, 0.0) = 0.50\\ \delta_{r2}(0.5, 1.0, 0.2) = 0.40\\ \delta_{r3}(0.5, 1.0, 0.0) = 0.50\\ \delta_{r4}(0.5, 1.0, 0.9) = 0.05\end{array}\right\} \rightarrow e1 : \langle r3, r1\rangle$ preferred

   We assume goal "p" can be attained. The values of the activity factors are $activity_{r3} = 0.5$, $activity_{r1} = 0.5$. Following damping factors are not equal to zero: $D_{r2} = 0.2$, $D_{r4} = 0.9$

2. $\left.\begin{array}{l}\delta_{r1}(0.5, 0.5, 0.0) = 0.25\\ \delta_{r2}(0.5, 1.0, 0.2) = 0.40\\ \delta_{r3}(0.5, 0.5, 0.0) = 0.25\\ \delta_{r4}(0.5, 1.0, 0.9) = 0.05\end{array}\right\} \rightarrow e2 : \langle r3, r2\rangle$ preferred

   We assume goal "p" cannot be attained. The values of the activity factors are $activity_{r3} = 0.75$, $activity_{r1} = 0.25$, $activity_{r2} = 0.5$. Following damping factors are not equal to zero: $D_{r2} = 0.4$, $D_{r3} = 0.2$, $D_{r4} = 0.9$

3. $\left.\begin{array}{l}\delta_{r1}(0.25, 0.75, 0.0) = 0.19\\ \delta_{r2}(0.50, 0.50, 0.4) = 0.15\\ \delta_{r3}(0.30, 0.25, 0.2) = 0.06\\ \delta_{r4}(0.50, 1.00, 0.9) = 0.05\end{array}\right\} \rightarrow e1 : \langle r3, r1\rangle$ preferred

   We assume goal "p" can be attained. The values of the activity factors are $activity_{r3} = 0.875$, $activity_{r1} = 0.625$, $activity_{r2} = 0.25$. In addition, following damping factors are not equal to zero: $D_{r2} = 0.4$, $D_{r4} = 0.9$ Note that r1 was chosen over r2 because of the damping factor.

4. $\left.\begin{array}{l}\delta_{r3}(0.2, 0.125, 0.0) = 0.025\\ \delta_{r4}(0.5, 1.000, 0.9) = 0.050\end{array}\right\} \rightarrow e3 : \langle r4\rangle$ preferred

   Finally, the system chooses execution three as the preferred one. Note that we omitted the calculation of δ_{r1} and δ_{r2} due to space reasons.

After presenting the main ideas of our approach, the next section presents an algorithm implementing our control system.

Algorithm Description

To complete the description of our approach, we include a discussion of an algorithm that implements the presented ideas. The algorithm uses a graph-based representation of the SKB, in particular of the included horn theory. Figure 1 already provided a sketch of how our system finds preferred executions. In this section we provide the details of the algorithm. We recommend looking at Figure 3 in order to receive a first impression of the algorithm.

As presented, a run of the SKB splits into following tasks:

1. Update the SKB, in particular any damping- and activity factors.
2. Calculate weights for every consequent.
3. Find a preferred execution.
4. Conduct the preferred execution.
5. Goto first item.

We split the discussion of our algorithm into pieces corresponding to the given enumeration and start by explaining how the SKB update works in our implementation.

- Update the SKB:
  At system start, all damping and activity factors ($activity_{rX}$ in the example) are set to zero. If the system already has finished a cycle as presented in the previous enumeration, activity factors are updated according to entailed consequents: The activity factor will be incremented for each consequent that is part of the preferred execution. For each consequent that has been successfully reached, because all actions worked out as expected, the damping factor (D_{rX}) is reduced if it was greater than zero. All consequents that did not become true as expected are penalized by increasing the damping factor by some predefined amount. After these factors have been

updated, all activity factors (including the ones from consequents not in the execution) get aged e.g., divided by two.

Apart from maintaining the damping and activity factors, the SKB also has to be cleaned from propositions calculated in order to attain a certain goal: Only if the system has successfully reached a goal, the corresponding propositions are removed. Propositions set by the environment are being left alone by the SKB update function as these get set and removed by some special driver layer responsible for mapping external world states to internal states. The reason why we need to remove propositions from memory was explained in Definition 8.

- Calculate weights:
  At system start, when both activity and damping factors are set to zero, the weights reflect the normal operating condition as specified by the knowledge engineer. During successive runs, due to the increment and aging operations, goals with γ-function maxima at higher activity values will be pursued more often than goals having a γ-function maximum at a lower activity value. In case a failure occurs, the system will gradually adapt its behavior, performing system reconfiguration by shifting weights toward working solutions. Weight is calculated as was shown in Example 2.
- Find and execute a preferred execution:
  As weights have been set up, finding the preferred execution reduces to a normal backtracking search as depicted in Figure 1. From the goal having the greatest weight, the system searches its way through the horn theory to the current system state. If it thereby encounters an alternative, it takes the branch with the highest weight. In case this branch turns out to be unsatisfiable, the system does a backtracking operation and takes the next-best alternative. After the system has found a goal that is satisfiable under the assumption that all actions are carried out without failure, it starts executing the found path. This is somewhat in contrast to the definition of a run of a preferred execution as the goals are not approached in a parallel manner. It is a simplification and a relaxation at the same time: It's a simplification because (a) it would have been more complex to implement parallelism on our prototype and (b) the selected rule set will – in most cases – be smaller than the set of the complete preferred execution. That said, our system does adhere to the spirit of the definition of preferred execution because after executing one goal, it selects another goal for execution until no goal can be attained anymore or no goal is left to process. After all goals were processed, the system will finish updating the SKB and start the evaluation cycle from the beginning.

Fault Localization

The control system described so far implicitly uses diagnosis for reconfiguration. In this section, diagnosis in terms of fault localization, i.e., finding the root cause of unexpected behavior, of the system using the SKB is presented. Fault localization can be conducted on two different layers of abstraction. One alternative is to work on the layer

```
function Main()
    SetAllFactorsZero();
    do forever
      CalculateWeightForConsequents();
      SortAndFilterGoals();
      foreach goal do
        search for execution e
          UpdateSKB(e,Execute(e));
      AgeActivityFactors();
      CleanPropositions();

function UpdateSKB(e,reachedGoal)
    for all rules r in e do
      IncreaseActivity(r);
      if reachedGoal == true
        if DampingFactor(r) > 0
          DecrementDampingFactor(r);
      else
        IncreaseDampingFactor(r);
```

Figure 3: Sketch of the employed algorithm. See Section "Algorithm Description" for details.

of actions. The other one is to use damping factors in order to obtain valid diagnoses. In this section we give an overview of how diagnosis can be included in the presented theory, what algorithms can be used to obtain diagnoses, we sketch advantages and disadvantages of each approach, and finally we present cases of how the gained information can be used within the presented theory. First, we discuss diagnosis based on the action layer.

Although not intended in the presented theory, it is possible – and actually implemented in our prototype – to store all actions that failed upon execution. Because we require all actions to be monitored when executed, the system immediately knows when an action fails. If the SKB includes a richer model of the system that provides structural information, in particular a set of components each action is dependent on, we are able to conduct a fault localization as presented by Reiter (Reiter 1987) in order to pin down the module not working according to our expectations. This knowledge could then be used to fine-tune preference criteria of consequents of rules.

We informally present an argument for the previous statement: Given a SKB that includes structural information about the actions and that is complete in a way so that all goals can be attained. Then each action a is dependent on a set of components $C_a = \{c_1 \dots c_n\}$. If the action cannot be carried out, then C_a naturally is a conflict set. Together with other conflict sets of failed actions, we can employ the hitting set algorithm of (Reiter 1987) and calculate diagnoses.

If we consider the standard benchmark of the SKB that stores information about failed actions, namely the damping factors, it is certainly possible to utilize the conflict-set based approach in order to calculate diagnoses. Having said that, there is evidence that this approach is not well suited. Because a rule may contain more than one action and the in-

```
// Small rule set, controlling the example device.
LowE <= Test(Battery.Low) | Test(Battery.Crit);
HaveGuesstimatePos <= Do(COMP.GetBearing)
       & Do(BARO.GetHeight) & Do(CLOCK.GetTime);
HaveGpsPos <= not_Test(GPS.Disabled)
       & Do(GPS.GetPosition);
HavePosition <= HaveGpsPos | HaveGuesstimatePos;
// Goals
Goal2:  <= HavePosition & Do(NF.SendPosition);
Goal1:  <= HavePosition & Do(GSM.SendPosition);
Powersave:  <= LowE & GPS.Disabled & GSM.Off
       & CPU.ClockDown;
SwitchOnGsmPwr:  <= not_LowE & Test(GPS.Disabled)
       & GPS.Off;
SaneShutdown:  <= Test(Battery.Crit)
       & ExampleSystem.Off;
Goal0:  <= not_Test(Battery.Crit) & ExampleSystem.On;
```

Figure 4: Example rule set written in our SKB-rep. language for illustration purposes only.

crease and decrease operations on damping factors are non-deterministic it seems that this method will provide coarse results only. One option to increase the resolution is to take activity factors and function γ into account. Damping factors can best be used to discriminate long staying faults from transient ones because the value of a damping factor reflects the amount of time a fault is present.

In a way, the two different methods amend each other: While conflict-set based diagnosis provides exact diagnoses candidates, damping factors are useful for checking and selecting proposed diagnoses because of the long-term memory they provide.

The information gained by employing diagnosis is best used when diagnoses can influence the SKB, in particular γ. Because of environmental interactions, however, this kind of feedback is not trivial to master and will require the SKB to include a model of the environment. Another interesting area of application of the discussed fault localization approaches is off-board diagnosis.

The main focus of our presented approach has been put on the control and reconfiguration aspect. We understand reconfiguration (or repair) to be one important task of diagnosis besides fault detection and fault localization, so we haven't implemented diagnosis in terms of root causes yet. The next section will give a quick overview of results obtained while working on a first implementation of the presented algorithm.

Results

We have implemented a version of our framework for a Microchip PIC 18F micro controller. Due to the limited memory size of 128K bytes program memory and less than 4K bytes random access memory of the device and the need to also include firmware for the driver layer, we were not able to implement every feature of the architecture presented in this paper. In particular, the implementation lacks support for arbitrary γ-functions, and there is no fault-localization diagnosis algorithm due to memory size constraints. Nonetheless, even this initial version of our approach proved its ability for adaptability and automated low-level repair.

Tables 1 and 2 present a case study and illustrate the capability of the system to adapt to changing environments. The rule set used during this experiment is shown in Figure 4 and contains some features not discussed in this paper. By only using damping factors and the inherent diversity within the rule set, the system was able to stick to its task: Generating a stream of position readings. Therefore, no low-level repair command had to be issued by the software system. Note that we did not discuss low-level repair in this paper. Finally, we want to point out that G1 and G2 are two redundant possibilities of reaching the abstract goal "send current position". Our system of course knows about this and treats G1 and G2 as alternatives of one high-level goal. For the sake of simplicity and clarity we will refer to G1 and G2 as if they were (distinct) goals.

The results of Table 1 are interesting to compare because they demonstrate the system is quicker in adapting to a single fault than to multiple ones which tend to cause a fuzzy response. Please note that our example implementation had a policy of sticking to GPS as long as possible and to ignore a certain amount of failures of this module. Therefore the system needs more time to adapt to the non-functional GPS module than it needs when adapting to a faulty GSM module. Due to space constraints we have skipped all steps after step number ten, although the experiment shows that, e.g., for run three the system stabilizes in choosing G2 over the non-functional G1.

Table 2 shows a more complex set of induced faults. Note again that the system has a policy of using GPS for a certain amount of time, even if it seems faulty. In general the experiment separates into three parts. The first part includes steps one to five. This part simulates transient, changing faults with multiple influences on goals. Take, for example, step number three: Because the system does not *know* that GPS and NF failed at the same time, it tries to reach goal two (that uses NF) in step four without success. If the system would have inferred that the GPS component was responsible for not reaching goal one in step three, the switch to goal two in step four would not have happened. This is a good example why our approach still can benefit from exact fault localization. The second part of the experiment is a sequence of GPS faults. As we know that the system has the policy of preferring GPS it comes as no surprise that the system can't reach a few goals in sequence. Again, the system does not infer *why* it was unable to reach the goals and keeps switching alternatives. So the first two parts of the experiment recommend to include techniques to reason about a fault. However, our prototype system (and the main field of application) is geared toward small devices and low energy consumption which is in contrast to our wish of expanding the presented approach significantly. The last part of the experiment finally shows how our system adapts to faults of the NF component: In run two, the system succeeds in seven

	Permanent Faults							
Step	Run 1	Faults	Run 2	Faults	Run 3	Faults	Run 4	Faults
1	G0, G1	-	G0, G1	-	G0, G1	-	G0, G1	-
2	G2, G0	-	G2, G0	-	¬G2, G0	GPS	¬G2, G0	GPS
3	G1, G0	-	G1, G0	-	¬G1, G0	-"-	¬G1, G0	-"-
4	G2, G0	GSM	¬G2, G0	GPS	G0, ¬G2	GPS, GSM	G0, ¬G2	GPS, GSM
5	¬G1, G0	-"-	¬G1, G0	-"-	G0, ¬G1	-"-	G0, ¬G1	-"-
6	G2, G0	-"-	G0, ¬G2	-"-	G0, ¬G2	-"-	G0, ¬G2	-"-
7	G2, G0	-"-	G0, ¬G1	-"-	G0, ¬G1	-"-	G0, ¬G1	-"-
8	G2, G0	-"-	G0, ¬G2	-"-	G0, G2	-"-	G0, G2	-"-
9	G2, G0	-"-	G0, G1	-"-	G0, ¬G1	-"-	G0, ¬G1	-"-
10	G2, G0	-"-	G0, G1	-"-	G0, G2	-"-	GP, G0, G2	GPS, GSM, low battery

Table 1: Steady Transient ("Permanent") Faults
This example demonstrates the goal-balancing and balanced action selection with GPS strongly preferred to fallback methods. "G1" sends acquired position data by GSM, while "G2" sends the information by some near-field communications link. "GP" does some power housekeeping in order to lower total power consumption (e.g. clock down). An attached "¬" means the goal failed. No low-level repair command was issued.

out of nine cases that have a working solution.

Before coming to related research, we want to say that the frequency specified by γ had been set to "always" for the abstract goal of transmitting data during the experiment. Therefore the system tried to reach either G1 or G2 in each step.

Related Research

Tremendous amount of research has been done in the area of automated system control. In principle, the beginnings date back to the very first planning systems, for example STRIPS (Nilsson & Fikes 1970). Planning algorithms in general try to solve the problem of finding a sequence of actions that, when applied in some start-state, lead to a specific target state.

A lot of planning systems were especially proposed in the context of AI controlled systems. Most of these planning systems include some sort of re-planning and fault detection capability.

The Livingstone architecture proposed by Williams and colleagues (Williams *et al.* 1998) was used aboard the space probe Deep Space One to detect failures in the probe's hardware and to recover from them. In contrast to Livingstone our system does not require to locate the real cause of a fault before recovering. Moreover, experiences gained when executing the SKB are directly used by our system to select the optimal actions. That said, it is also possible in our approach to gain information regarding the fault by considering actions which often fail.

In (Dearden & Clancy 2002) and (Verma *et al.* 2004) particle filter techniques were used to estimate the state of the robot and its environment. These estimations together with a model of the robot were used to detect faults. The most probable state is derived from unreliable measurements. The advantage of this approach is that it is able to handle non-Gaussian uncertainties of the robot's sensing and acting as well as uncertainties in its environment. In our approach we do not need to handle probabilities directly. Although our approach depends on factors somewhat related to probabilities, the selection of the current behavior tries to minimize a given gradient and does not directly use these factors.

Inspired by biological systems, a different view to intelligent behavior was born: "Intelligence is in the eye of the observer" (Brooks 1991) was the new line of reasoning. Basically the statement means that only the observer decides whether a certain behavior is intelligent or not. It's thereby unimportant whether the seemingly intelligent behavior was generated by very complex calculations or merely by a subsumption of different, very simple basic behaviors. Probably the biggest technical innovation of this new concept at that time was the turning away from computationally expensive, mostly non concurrent, rarely reactive planning engines. Behavior is now created by concurrent, light weight, highly reactive, decentralized components ("behaviors"). Complex behavior is defined as a superposition of several simple behaviors.

Based on the same basic idea, a lot of different versions of behavioral systems were published: Among them are the subsumption architecture from Brooks (Brooks 1991), and the command fusion architecture from Rosenblatt and Payton (Rosenblatt & Payton 1989). Teleo-reactive programs (Nilsson 1994) present a somewhat special case, because TR-programs can be used to execute STRIPS generated plans.

Most of these approaches include some inherent robustness regarding to faults. Either by providing enough redundancy in hardware or by specially designed control algorithms.

Conclusion

Our approach is neither a conventional planning system with diagnosis capabilities, nor a behavioral system in view of the subsumption architecture. It's best compared to TR-programs (Nilsson 1994).

There are some differences though. The biggest difference lies in action selection and real-time capability: Our approach relies on dynamic gradients and discrete time steps without guaranteed real time capabilities, while TR-

	Transient Faults			
Step	Faults	Run 1	Faults	Run 2
1	GSM	G0, ¬G1	GSM	G0, ¬G1
2	NF	¬G2, G0	NF	¬G2, G0
3	NF, GPS	G0, G1	NF, GPS	G0, G1
4	NF	G0, ¬G2	NF	G0, ¬G2
5		G0, G1		G0, G1
6		G0, G1	GPS, GSM	G0, ¬G1
7	GPS	G0, ¬G1	GPS, GSM	G0, ¬G2
8	GPS	G0, ¬G2	GPS	G0, ¬G1
9	GPS, GSM	G0, ¬G1	GPS	G0, ¬G2
10	GSM	G0, G2	GPS, GSM	G0, ¬G1
11		G0, G1	GSM	G0, G2 (pos wo. GPS)
12		G0, G1		G0, G1 (pos wo. GPS)
13	NF	G0, ¬G2	NF	G0, ¬G2 (pos wo. GPS)
14	NF	G0, G1	NF	G0, G1 (pos of step 13)
15		G0, G1	NF	G0, G1 (pos wo. GPS)
16		G0, G1	NF, GPS	G0, ¬G1
17	Compass	G0, G1	NF, Compass, GSM	G0, ¬G2
18		G0, G1	NF, Compass	G0, G1 (pos of step 17)
19		G0, G1	NF	G0, G1
20		G0, G1	NF	G0, G1

Table 2: Transient Faults. "G1" sends acquired position data by GSM, while "G2" sends the information by some near-field communications link. An attached "¬" means the goal failed. See also Table 1.

programs (consisting of TR-sequences) have a fixed priority order and real-time, circuit semantics. TR-programs also feature durative actions while our approach is based on finite actions we need in order to simplify diagnosis. Nevertheless it is entirely possible to extend our approach to cope with durative actions by e.g., introducing new predicates "StartAction(<action>)" and "StopAction(<action>)".
TR-programs always evaluate all conditions. This is different to our approach, as we have discrete evaluation points.
It is possible to hierarchically compose TR-sequences. While we cannot directly compose SKBs, we can compose a system that comprises several runtime systems with SKB.

The most important properties of our approach solving the problem of autonomous system control and repair are the following:

- Declarative Knowledge Base
- Integrated Diagnosis with Reconfiguration and Low-level Repair
- Fault Localization

Mainly two areas are left for future research: On the one hand, it is not entirely clear how probability based methodologies relate to our system in detail. An in-depth comparison between the two approaches is necessary in order to highlight the differences. On the other hand, we want to develop a methodology for the creation of the system knowledge base that is used by our control system.

Acknowledgments

This work has been supported by the FIT-IT research project "Self Properties In Autonomous Systems (SEPIAS)".

References

Brooks, R. A. 1991. Intelligence without reason. In Myopoulos, J., and Reiter, R., eds., *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, 569–595. Sydney, Australia: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Dearden, R., and Clancy, D. 2002. Particle filters for real-time fault detection in planetary rovers. In *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis*, 1–6.

Nilsson, N. J., and Fikes, R. E. 1970. Strips: A new approach to the application of theorem proving to problem solving. Technical report, Stanford Research Institute.

Nilsson, N. J. 1994. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* (1):139–158.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32:57–95.

Rosenblatt, J. K., and Payton, D. W. 1989. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proc of the IEEE Int. Conf. on Neural Networks*, volume 2, 317–324. Washington, DC: IEEE Press.

Verma, V.; Gordon, G.; Simmons, R.; and Thrun, S. 2004. Real-time fault diagnosis [robot fault diagnosis]. *Robotics & Automation Magazine* 11(2):56 – 66.

Williams, B. C.; Muscettola, N.; Nayak, P. P.; and Pell, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–47.

Towards a Modeling Approach of Dynamic Systems for a Multi Model Based Diagnosis

Emilie Masse and Marc Le Goc

Laboratoire des Sciences de l'Information et des Systèmes - LSIS
UMR CNRS 6168 - Université Paul Cézanne Aix-Marseille III
Avenue Escadrille Normandie Niemen13397 Marseille Cedex 20 – France
{emilie.masse, marc.legoc}@lsis.org

Abstract

This paper proposes to combine the expert assumptions and a multi model approach to model a dynamic system for a diagnosis task. The idea is that the experts, implicitly, define a level of abstraction to model the system to diagnose such that the computational problem of the diagnosis is minimized. The aim of our works is then to define a methodology to build a coherent set of models of a dynamic system that uses the experts' assumptions. This methodology is based on the use of a conceptual model of reasoning to interpret the experts' knowledge and building the underlying models of the dynamic system to diagnose: a structural model and a functional model that are compatible with Reiter's theory and a behavioural model. The methodology and the modeling formalisms are presented on the technical diagnosis of a simplified car. Then, an application to a real world dynamic system, the Cublize dam, is proposed to show the operational aspect of our approach.

Introduction

Lot of works in the domain of Artificial Intelligence in general, and in the field of model based diagnosis in particular, are concerned with the design of knowledge based systems to supervise, diagnose and control industrial process [Basseville and al, 1996]. One of the main problems of this matter comes from the dynamic aspect of industrial processes, which poses the problem of the acquisition and the representation of the underlying temporal knowledge. This problem is difficult because the temporal knowledge is often mixed with other types of knowledge. This observation has motivated the works about the multi model based diagnosis, which emphasis the separation of the different types of knowledge in different models. The aim is to simplify both the knowledge representation to build coherent models and the reasoning on the different models. But one of the remaining problems is to define the adequate level of abstraction to build models that are humanly understandable and technically efficient for a diagnosis task in the meaning of Reiter.

In this paper, we propose to design models at the same abstraction level than the experts to allow an efficient diagnosis task. To this aim, we propose the basis of a method to build four models. The first one is a conceptual model of the problem solving method the experts use in the diagnosis task. The second one is a structural model containing the knowledge about the components and the structures of the system to diagnose. The third one is a behavioural model containing the knowledge about the states and the events that control the evolutions of the system. The last model is a functional model containing the variables and the functions that defines the values of the variables. Our goal being the modelling of dynamic systems for diagnosis, the emphasis is made on the behavioural model.

So, section 2 positions our approach according to the main modelling approaches for diagnosis. Section 3 presents the basis of our methodology through the application of the modelling formalisms and the modelling procedure to a toy problem: the technical diagnosis of a car. An operational application on the Cublize dam from the expertise of the Cemagref is provided in section 4. Finally, section 5 states our conclusions and perspectives.

Main Diagnosis approaches

The diagnosis approaches described in the literature can be split in three main approaches. The first one is the heuristic approach [Clancey, 1985] where experts' knowledge (heuristics) relies on empirical associational relations of the form: symptoms → faults (diseases). This approach supposes that this kind of knowledge can be elicited from experts and also be represented using adequate knowledge representation languages. This approach is the basis of the experts' systems and suffers for the same limitations and problems.

The limitations and the problems of the experts' systems approach has motivated the Model Based Diagnosis approach (MBD, [Reiter, 1987]) where the knowledge about the system is represented in a unique logical model.

In this approach, the system to diagnose is a pair (SD, COMPS) where SD, the description of the system, is a set of formulae of the first order predicate logic with equality and COMPS, the system components, is a finite set of constants. An observed system is a triplet (SD, COMPS, OBS) where OBS is a set of first order formulae with equality. When an inconsistency is detected in the theory (SD, COMPS, OBS), a diagnosis is looked for in the terms of the minimal sets of elements of COMPS of which the functioning is abnormal. A diagnosis for (SD, COMPS, OBS) is then a first order formulae $D(\Delta)$ with $\Delta \subseteq$ COMPS such that SD $\cup$ OBS $\cup$ $D(\Delta)$ is satisfied. $D(\Delta)$ is a conjunction of AB(x) predicates, $x \in \Delta$, meaning "component x may behaves abnormally" (hypothesis). The Model Based Diagnosis poses two main problems. The first is the source of the model (SD, COMPS). The used models are often design models in which the number of components can introduce difficulties to compute $D(\Delta)$ (exponential explosion of hypothesis). This problem is crucial when diagnosing systems are made with a large amount of components. The second major problem is that the observations in OBS are not dated. Then, The temporal knowledge can not be taken into account in the computation of $D(\Delta)$.

So, The Multi Model Based Diagnosis (MMBD) has been proposed to avoid these problems (cf. [Zouaoui, 1998] and [Thetiot, 1999] for examples). This approach defines four models to describe the system to diagnose [Chittaro et al., 1993]. The first one is the Structural Model that describes the components and their connexions. The structural model links components with physical phenomena. The second model is the Behavioural Model describing the operations that are implemented in the components. This model describes the mechanisms that control the temporal evolutions of the variables. The third one is the Functional Model describing the functions allowing reaching an objective. These functions are the results of the behaviour of the components over time. The last model is the Teleological Model describing the goals assigned to the system. This model allows the definition of the unsuitable uses of the system.

According to [Zouaoui, 1998], the qualitative modelling of complex systems is not sufficient because it concerns with only the structural and the behavioural models. Adding a functional model is a way to reduce the number of hypothesis (candidates) and so, minimizing the computational of the diagnosis cost. Moreover it improves the understanding of the system behaviour. The MMBD proposes a structural model which represents the whole system and a functional model which is used to guide the problem solving process. The MMBD is then adapted to diagnose complex systems [Thetiot, 1999].

Multi Model Based Diagnosis with Expert Assumptions

Indeed, the MMBD is still concerned with the computational problem of the diagnosis as the dimension of the solution space still evolves as an exponential with the number of the components declared in the structural model (Zouaoui, 1998). This problem is directly connected to the abstraction level used to model the knowledge. Where the MMBD generally consider that this level of abstraction is defined by the subjective decision of the designer(s), a focusing mechanism is then a necessity to have an efficient diagnosis algorithm.

Our works are based on the hypothesis that an expert uses a set of models at a level of abstraction that allows efficient diagnosis reasoning and this level of abstraction differs from the one of the designer. The problem is that the experts formulate their knowledge about the system, not the models of the system they use to formulate their knowledge.

Our works about dynamic system modeling have some similarities with the approach presented in (Traoré, 2003), with a fundamental difference however: considering that a model of a system depends of the problem to solve, we include the interpretation knowledge and the fundamental knowledge in the same set of models representing a system. And as the Systemic Approach (Le Moigne, 1999), we apply the knowledge separation principle: distinguishing the knowledge with different nature in different models.

We propose to use a CommonKads conceptual model to represent the interpretation knowledge (Schreiber and al, 2000). Such a conceptual model is an instanciation of a CommonKads template of a cognitive task. This model is then used to interpret the knowledge about a system with the aim of building three models (Zanni and al, 2005): a structural model describing the relations between the components of the system, a functional model describing the relations between the values the variables of the system can take (i.e. mathematical functions) and a behavioural model describing the states of the system (corresponding to the operating modes of (Chittaro and al, 1993)) and the discrete events firing the state transitions.

The relation between these models is made with the notion of variable: a variable used in a function of the functional model is associated with an element of the structural model and a discrete event is defined as the affectation of a value to a variable.

Modeling Principles and Formalisms

We define a model as an organized set of binary relations B(Sl, Sr) between symbols Si denoting objects described in the domain ontology of the conceptual model of the knowledge. A binary relation $B(S_i, S_j)$ is also denoted with a symbol (i.e. "*B*"). Such a binary relation only means that there exists a link between the objects denoted with the symbols S_i and S_j. A typical graphical representation of a binary relation is provided in Figure 1, where nodes are boxes or circles and the relation is eventually represented with an ellipse. When useful, the arcs can be oriented to show the orientation of a flow like typically a flow of energy, material or information.

Figure 1: Basic Binary Relation Template.

A model can then be represented with a graph where nodes are symbols and arcs are relations. Basically, a node symbol denotes a component or an aggregate of components (i.e. a structure) in a structural model, a variable in a functional model or a state in a behavioural model. An arc represents a link between two nodes. Such a link can be a connection link between two structures in a structural model, an information link between the values of variables in a functional model or a transition between two states in a behavioural model. Consequently, a binary relation in a structural model represents a connection between two structures; a mathematical function in a functional model and a transition between two states in a behavioural model. It is to note that a set of binary relations with the same right node symbol can be aggregated in a single n-ary relation when this aggregation is meaningful (an arithmetical function between 3 variables for example).

To illustrate the modeling process, let us take the example of the (simple) knowledge base used to diagnose a car according to (Schreiber et al, 2000). Figure 2 proposes a graphical representation of the set $R = \{R_i(P_c, P_e)\}$ of nine rules constituting the knowledge base. In this graph, a rule $R_i(P_c, P_e)$ denote a logical consequence relation from a proposition P_c to another P_e. This logical relation is used to represent a causal relation between a cause ("Fuel Tank empty" for example) and an effect ("Gas in engine false" for example). We will use this simple knowledge base to show the way our formalism can be used to build the underlying structural, functional and behavioural models the experts uses to formulate this knowledge.

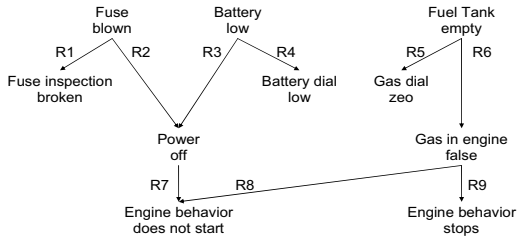


Figure 2: An example of a knowledge base.

To this aim, we will use the classical and minimal CommonKads diagnosis template «hypothesis generation and hypothesis discrimination» (Zanni and al, 2005). This template (Figure 3) considers the diagnosis reasoning as being made of two basic inferences, one to generate the hypothesis from the observed behaviour and the second to discriminate between the different hypotheses according to the observed behaviour. This template allows the classification of the propositions contained in the knowledge base in a set of observed behaviour (Fuse inspection broken, Engine behaviour does not start, engine behaviour stops, Battery dial low and Gas dial zero) and a set of hypothesis (Fuse blow, Battery low, Fuel Tank empty, Power off and Gas in engine false). The observed behaviour contains the set of complains that motivates the diagnosis reasoning (Engine behaviour does not start, engine behaviour stops). This functional classification of the propositions leads to distinguish a first set of rules $S_1=\{R_1, R_4, R_5\}$ concerning the unobservable states of the car from the second set of rules $S_2=\{R_2, R_3, R_6, R_7, R_8, R_9\}$ that expresses the propagation of an unobservable car state (Fuse blow, Battery low, Fuel Tank empty) to another unobservable car state (Power off, Gas in engine false) and finally, to the complains.

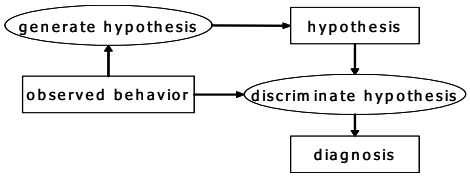


Figure 3: The Diagnosis Template

This simple functional analysis of the set of rules *R* allows the interpretation of the structure of the propositions under the form of binary relations between a variable (Fuel Tank for example) and a value (empty): each proposition corresponds to a predicate *Equal(Variable, Value)*. Consequently, a rule is an instantiation of a relation of the form $Cause(x_i=v_1, x_j=v_2)$ where the symbol "=" denotes the predicate "*Equal*", "*x*" denotes a variable and "*v*" a value. The set of variables can then be build, and when associating a component to each variable, the set C of components is then defined.

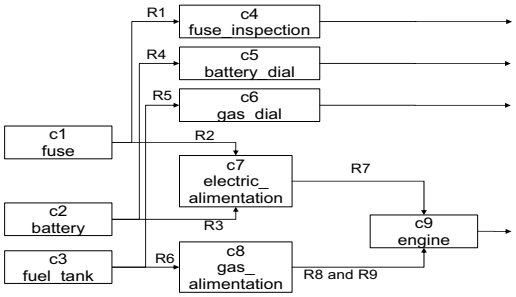


Figure 4: Structural model.

The relation $Cause(x_i=v_1, x_j=v_2)$ supposes that there exists a physical relation between the variables x_i and x_j that is to say between the components or the aggregates c_i and c_j the variable *xi* and *xj* are linked with. The set of rules can then be interpreted as connection relations of the form $ConnectedTo(c_i, c_j)$ and represented in the structural model of Figure 4. To this aim, the nodes of the basic binary relation template of figure 1 are represented with rectangles and the relation symbol "*Ri*" refers to the rule symbol of the knowledge base. The symbol components "electric_alimentation" and "gas_alimentation", respectively associated with the variables "Power" and

"Gas_in_engine", denote abstract aggregates of components.
When denoting $X=\{x_i\}$ the set of symbol variable associated with the set $C=\{c_i\}$ of symbol components, the knowledge base R can be rewritten as follow:

- R1: If x_1=Blown Then x_4=Broken
- R2: If x_1=Blown Then x_7=Off
- R3: If x_2=Low Then x_7=Off
- R4: If x_2=Low Then x_5=Low
- R5: If x_3=Empty Then x_6=Zero
- R6: If x_3=Empty Then x_8=False
- R7: If x_7=Off Then x_9=Does_not_start
- R8: If x_8=False Then x_9=Does_not_start
- R9: If x_8=False Then x_9=Stops

The "If P Then Q" form refers to the application of the Modus Ponens with the fact "P" and a rule of the form "$P \Rightarrow Q$". The relation $Cause(x_i=v_1, x_j=v_2)$ means that there exists a logical relation between the fact "$x_i=v_1$" and "$x_j=v_2$". The set of rules can then be interpreted as a set of relations of the form $(x_i=v_1)\Rightarrow(x_j=v_2)$. When defining the domain set of value for each variables, each relation of the form $(x_i=v_1)\Rightarrow(x_j=v_2)$ subsumes a function of the form $x_j=f_{ij}(x_i)$ at least defined when $x_i=v_1$ and $x_j=v_2$. When two functions $f_{ij}(x_i, x_j)$ and $f_{kj}(x_k, x_j)$ share the same output variable x_j , a new function $f_j(x_i, x_k, x_j)$" can be defined.

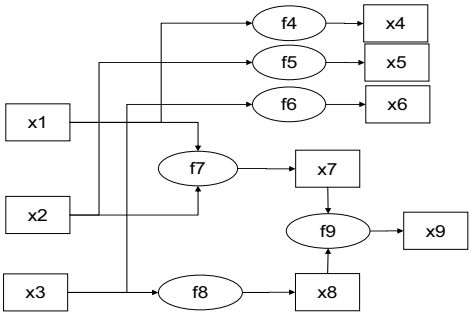


Figure 5: Functional model.

The functional model of Figure 5 is deduced from the set of rules. This model is compatible with the structural model in the sense that (i) each variable xi is associated with an element ci of the component set C and (ii) each function $f_{ij}(x_i, x_j)$ is linked with a connection between two elements c_i and c_j of the structural model. In this functional model, the nodes are represented with rectangles and the relations are represented with arcs and ellipses in which the symbol relation f_i is written. This means that rectangles specify variable names and ellipses specify function names.
When the value set of each variable xi can be defined, the set $F=\{f_i\}$ of functions f_i can be entirely specified with tables of values. The set of rules R does not define all the values a variable can take in the car system. But, it is always possible to define the value set of each variable with the definition of the complement value of the defined values. For example, the value set of the variable $x1$ is {Blown, not_Blown}. In this example, the value set of the variables except x_9 is isomorphic with the Boolean set $B=\{V, \neg V\}$. This leads to the tables of figure 6. These tables are formulated independently of the variables, when using the notation $o(f)$ and $i(f)$ to denote the value of the output and the input of the function "f".

o(f6)=f6(i(f6))	
i(f6)	o(f6)
Empty	Zero
Not_Empty	Not_Zero

o(f8)=f8(i(f8))	
i(f8)	o(f8)
Empty	False
not_empty	True

o(f7)= f7(i1(f7), i2(f7))		
i1(c7)	i2(f7)	o(f7)
not_blown	not_low	On
not_blown	Low	Off
Blown	not_low	Off
Blown	Low	Off

o(f9)= f9(i1(f9), i2(f9))		
i1(f9)	i2(f9)	o(f9)
On	True	Start
On	False	Does_not_start
Off	True	Does_not_start
Off	False	Does_not_start

Figure 6: Specification of the functions.

To introduce the behavioural model, let us consider the variable x_9 associated with the aggregate c_9 (i.e. the "engine"). The value set of x_9 is {Off, On} where the symbol "Off" means either "stops" or "does_not_start". So, the value "On" corresponds to some thing like "working ≡ not_stops" and "starts ≡ not_does_not_start". The complains corresponding to the predicate $x_9=Off$ can be interpreted as an undesirable car state (the car stops or the car does not start) and so, the predicate $x_9=On$ corresponds to the desirable car state "the car is working and starts".
According to the set of rules R, the car stays in the state "working" until the occurrence of a discrete event "No_power" (i.e. $x_7=Off$) or "No_gas" (i.e. $x_8=False$). In this case, the car transits from the state "Working" to a state "Out of Power" or "Out of Gas", which are by definition two transient states. As soon as the inertia will have no effect, the car will stop in a "Failure" state. When the ignition key will be off, the car will be in a "Broken down" state in which the predicate $x_9=Off$ is true: a repairing action is required to bring back the car in a normal "Stop" state. This analysis leads to the behavioural model of Figure 7, which is a finite state machine represented with the DEVS formalism (Le Goc et al, 2006). Nodes are states represented with circles and state transitions are the links between two states. Two types of transitions are distinguished: transitions conditioned with the occurrence of one discrete event as for example the event "No_gas" between the states "Working" and "Out of Gas", and autonomous transitions, represented with dashed lines, as for example between the state "Out of Gas" and the state "Failure". An autonomous transition is fired when the elapse time in a state q_i is greater than the maximum duration ΔT_i of the q_i state.

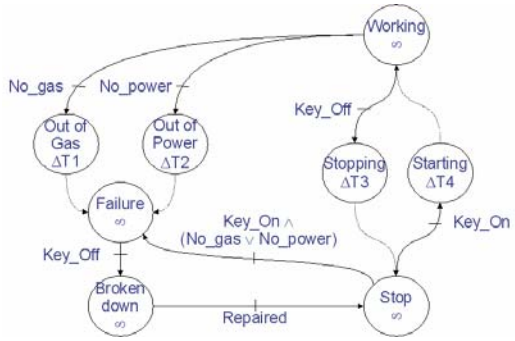


Figure 7: Behavioural model

A transition is characterized with a transition function δ so that: $Q(k+1) = \delta(Q(k), o(k+1))$, where $Q(k+1)$ is the state q_i at time t_k and $o(k+1)$ is the occurrence of the discrete event defined with the transition between q_i and $q_j = Q(k+1)$. According to the spatial discretization principle of (Bouché and al, 2005), a discrete event occurrence is a triplet (t_k, x, i) where x is the name of a variable, i is a discrete value and t_k is a time of the occurrence so that: $x(t_k) = i$. This interpretation allows linking the behavioural model with the functional model and so, with the structural model.

The models of Figures 4, 5, 6 and 7 are implicitly contained in the initial knowledge base, but the behavioural model is the most "hidden". Such an observation is very frequent because the dynamic properties are generally misunderstood. This is particularly clear when considering the absence of symbol denoting the ignition key of the car: the difference between the values "stops" and "does not start" becomes clear with the behavioural model of the car.

It is to note also that the behavioural model is made of two parts: one part (at the right of figure 8) describes the normal working of the car (Working, Stop, Starting and Stopping states), and the second part (at the left of figure 8) describes the abnormal working of the car (Out of gas, Out of power, Failure and Broken down states). Such a notion can not be defined with the functional model because it is defined as an organized set of relations between values of variables. This means that when considering dynamic systems, the normal behavioural model notion of Reiter's diagnosis theory for static systems has been shifted in the classification of the system states in normal and abnormal categories. This leads to design a new algorithm for diagnosing dynamic systems.

Coherence of the models

The modelling process is controlled with the requirement of keeping the consistency between the three models: any piece of knowledge must be projected into one of the three models and must not introduce any inconsistency in the other models. Otherwise, new knowledge will have to be obtained from the experts so as to raise the incoherence. This process is repeated until the end of the modelling. Two conditions have been identified to stop the modelling: an unsolved inconsistency and the incompleteness of the knowledge. The multi model approach relies on the invariance of the relation between the variables in the three models (Figure 8). The timed observations link the events to the variables of the functional model. So, the behavioural model and functional model are connected. And the functions described in the functional model being implemented with components, the variables are linked with the elements of the structural model. The notion of variable establishes then a link between the three models.

So the consistency is mainly controlled with the functional model. Thus, the functional model is the "pivot" between behavioural model and structural model.

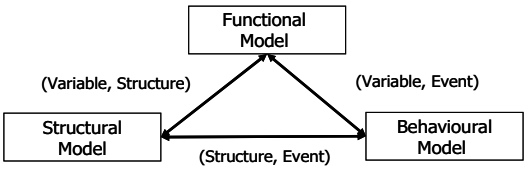


Figure 8: Links between the three models

The structural and functional models of Figures 4, 5 and 6 can be used in Reiter's diagnosis theory. The set of first order formulae (SD, COMPS) corresponds to a simple logical transcription of these models in the first order predicate logic. The set of components COMPS is deduced from the structural model: COMPS={c1, c2, c3, c4, c5, c6, c7, c8}

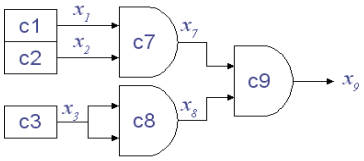


Figure 9: The knowledge as a logical circuit

The system description SD is deduced from the functional model when associating each function f_i with the component c_i corresponding to the output variable of the function f_i. For example, the function f_7 is associated with the component c_7 through the variable x_7: $x_7=o(f_7)$. Next, the symbols used to specify the functions f_i must be represented with the Boolean symbols "*T*" for "True" and "*F*" for "False". For example, when rewritten the symbols "not_Blown", "not_Low" and "Off" as "*T*" and the symbols "Blown", "Low" and "Off" as "*F*", the function f_7 become the logical function "AND" (Figure 9). The same is true for the functions f_8 and f_9. Finally, when considering the components c_4, c_5 and c_6 as sensors that can not failed, this lead to consider the following logical circuit of Figure 10. The system description is then:

SD = {
¬AN(x)∧FUSE(x) ⇒ o(x)=Not_blown,
¬AN(x)∧BATTERY(x) ⇒ o(x)=Not_low,
¬AN(x)∧FUEL_TANK(x) ⇒ o(x)=Not_Empty,
¬AN(x)∧AND_GATE(x) ⇒ o(x)=AND(i1(x), i2(x)),

```
//Component type declaration
FUSE(c1), BATTERY(c2), FUEL_TANK(c3),
AND_GATE(c7), AND_GATE(c8), AND_GATE (c9),
//Connexions
o(c1)=i1(c7), o(c2)=i2(c7), o(c3)=i1(c8), o(c3)=i2(c8),
o(c7)=i1(c9), o(c8)=i2(c9)
}
```

This shows that the modelling approach proposed in this section allows designing models that are compatible with the Model Based Diagnosis and that are formulated at the same level of abstraction of the experts. When supposing that this level of abstraction allows an efficient diagnosis, the resulting models would lead to an uncompleted but efficient diagnosis.

The application in the next section shows that this approach can be applied to a mainly dynamic process where the knowledge is mainly contained in a behavioural model.

Application

In this application, the knowledge base is the PHD manuscript chapter that Peyras, a dam expert of the Cemagref center of Aix-En-Provence (France), has devoted to the diagnosis of the Cublize's Dam story [Peyras, 2003]. This French dam suffered from internal erosion for about 10 years until its emergency reparation in 1989 (Figure 10). It is a homogeneous earth dam built of granite arena on a granite foundation, 16m high, and impounding a $2hm^3$ lake. It is made with a vertical drain the top of which is 2m lower than the normal water level and a horizontal drain at the foundation interface over the half downstream of the dam. The highest sections are instrumented with pressure sensors (C3, C4, C5) and the vertical-drain discharge is recorded (D1). The first reservoir filling began in November 1978 and since the lake level has remained close to the normal water level. In September 1988, a very wetland was observed at the downstream toe for the fist time. So, close monitoring was recommended. In mid-October 1988, the wetland grew larger and localized slides could be seen over a 10m length of the dam below the downstream berm. Two days later, muddy water carried at these places, and an emergency decision was made to completely empty the reservoir.

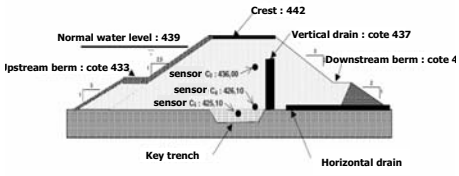


Figure 10: The section of Cublize dam

The investigations about the Cublize dam concluded that a mechanism of internal erosion was operative. The particle size grading of the fill material made it particularly sensitive to internal erosion, and this led to the gradual clogging of the vertical drain, aided by the fact that the drain did not meet standard filter rules. Clogging caused first the upstream fill to become gradually saturated and then the downstream fill as the infiltration water overtopped the drain. Saturation of the downstream fill material diminished its engineering properties, leading to shallow slides. The flowing water started to carry away the fines to the point where piping would have quickly developed and caused complete dam failure if drawdown had not been ordered quickly.

This story and the included diagnosis are explained through the scenario of figure 11 as given by Peyras.

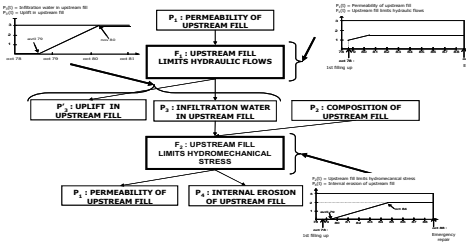


Figure 11: Part of the Cublize Dam Story

The first step of the knowledge analysis aims at preparing the knowledge for the projection in the three defined models. The conceptual model used to interpret the knowledge is the Sachem system's one [Le Goc, 2004]. This choice is justified by the fact that this model has been designed to model the diagnosis task of dynamic processes. The section of Cublize dam of the Figure 10 leads to the structural model of Figure 12.

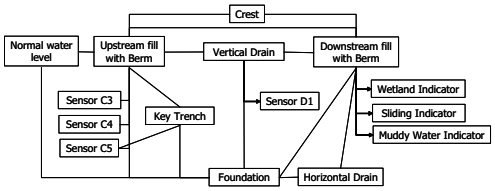


Figure 12: Structural model

The Figure 11 is a translation of the Cublize dam story under the form of a chronogram of phenomena occurrences (Figure 13).

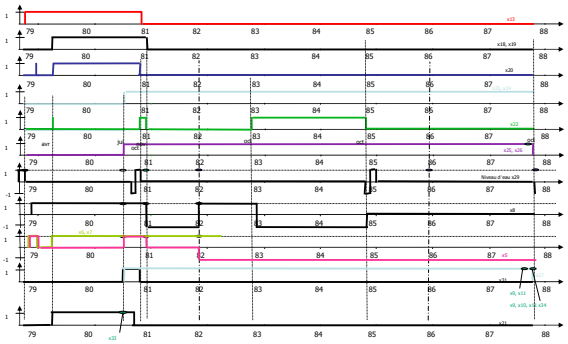


Figure 13: Cublize Dam Damage Chronogram

According to the Sachem conceptual model, a process phenomenon (PhPr) is a physical or a chemical phenomenon that governs the material or energy flow in the process. A process phenomenon occurrence modifies the evolution of signals (signal phenomenon, PhSg) provided by specific sensors corresponding to a variable. A process phenomenon is then associated with at least a variable trough a signal phenomenon. This means that the chronogram of Figure 13 specifies a sequence $\omega=\{o_k\}$ of discrete event occurrences of the form $o_k \equiv (t_k, x, i)$, each of them defining the start time or the end time of the phenomena of Figure 13:

ω = {(oct78, x8, increase in the water level), (nov78, x1, 436), (nov78, x2, 427.5), (nov78, x3, 427.5), (juil80, x5, 0.75), (dec78, x8, 439), (dec78, x1, 436), (dec78, x2, 428), (dec78, x3, 430), (apr79, x2, 430.5), (apr79, x3, 431.5), (nov80, x1, 437.5), (nov80, x1, 440), (nov80, x4, 1), (oct81, x8, 439), (nov81, x8, 437), (nov81, x1, 441), (nov81, x4, 1.5), (dec81, x8, 439), (oct82, x4, 1.2), (oct84, x4, 0.2), (nov85, x8, 439), (dec85, x8, 437), (mar86, x8, 439), (sept88, x5, wetland on the down fill), (sept88, x5, water on the down fill), (oct88, x5, expanded wetland on the down fill), (15oct88, x6, sliding), (17oct88, x7, muddy water), (dec88, x8, 433)}.

This sequence describes the story of the Cublize dam degradation. Eight variables are identified but the x_8 variable corresponding to the level of water in the dam, its values are the results of the dam state and can be "left out" to model the behaviour of the dam. We define a state transition as the modification from a normal value to an abnormal value of a variable and inversely. The knowledge base defines typically the value "0" as a normal value, defining consequently each value greater than "0" as an abnormal value. The sequence ω defines then the state transitions sequence of Figure 14: a state is then associated with a set of process phenomenon occurrences.

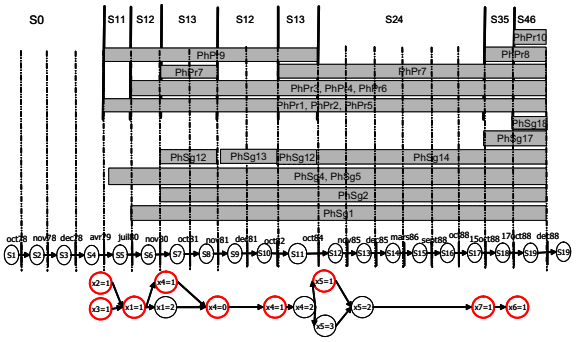


Figure 14: Behavioural model instantiation

This leads to the behavioural model of Figure 15. This is a DEVS model where the state transitions are defined with discrete events corresponding to the modification of the values of the variables x_1 to x_7. Naturally, this is not the complete behavioural model of Cublize dam but the part corresponding to the degradation story that is provided by the PHD chapter of Peyras (cf. Figure 11). It corresponds then to an abnormal behaviour, the normal behaviour being either implicit or not described in the PHD chapter. This model is a simplified version of the model validated by the dam's Experts of the Cemagref [Masse and al., 2007]. But we claim that this model structure the analysis of the degradation story of Cublize dam.

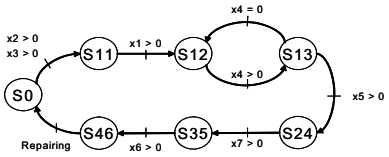


Figure 15: Behavioural Model

It is now possible to deduce a functional model from the behavioural model of Figure 15. The state transitions being defined as the modification of the values of at least one variable, the values of all the variables can be deduced from the behavioural model (Figure 16).

	x1	x2	x3	x4	x5	x6	x7
S0	0	0	0	0	0	0	0
S11	0	1	1	0	0	0	0
S12	1	1	1	0	0	0	0
S13	2	1	1	1	0	0	0
S24	2	1	1	1	1	0	0
S35	2	1	1	1	1	0	1
S46	2	1	1	1	1	1	1

Figure 16: Variable Values Evolution

When the process is in a particular state, the associated process phenomenon occurrences cause the evolution of the values of the variables. Some of these evolutions can be concerned with a discrete event that causes a new state transition. Consequently, in a first analysis, a function can be associated with each state making the relation between the values of the variables associated with the discrete events of the state transitions associated with the considered state. This principle leads to the functional model of Figure 17.

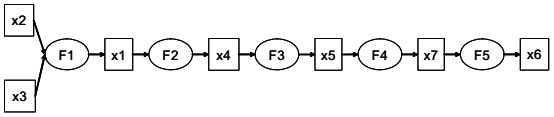


Figure 17: Functional Model

A function is defined with a value table representing its definition domain (Figure 18).

x1 = f1(x2, x3)		
x2	x3	x1
< 429	<430	< Drain height
< 429	≥ 430	∅
≥ 429	<430	∅
≥ 429	≥ 430	≥ Drain height

x4 = f2(x1)	
x1	x4
< Drain height	Not clog
≥ Drain height	Clog

x5 = f3(x4)	
x4	x5
Not clog	Not wetland
Clog	Wetland or water

x7 = f4(x5)	
x5	x
Not wetland	Not Sliding
Wetland or water	Sliding

x6 = f5(x7)	
x7	x6
Not sliding	Not muddy water
Sliding	Muddy water

Figure 18: Value tables

Naturally, this functional model can not be complete because the behavioural model is not complete. But such a

functional model shows the main numerical values the expert reasoning is based on and makes easier the knowledge acquisition and validation. We are now working to structure this model by the mean of the tetrahedron of Paynter (i.e. the T.O.S. of [Chittaro and al, 1993]).

The structural model of figure 12, the behavioural model of figure 15 and the functional model of figures 17 and 18 constitute the underlying knowledge the experts use to diagnose the Cublize' dam degradation. These models result of a systematic application of the principles presented in this paper. These principles are basis of a method that combine the knowledge acquisition phase about a diagnosis task and the modelling of the system that corresponds and legitimates the diagnosis knowledge. The formalisms of the functional and the structural models are compatible with the Reiter's logical theory of diagnosis. However, the Cublize dam shows that the behavioural model contains the fundamental knowledge the diagnosis reasoning is based on. And this model is not taken into account by the Reiter's theory.

Conclusion

This paper presents the methodological principles to acquire and model the diagnosis knowledge and corresponding dynamic system model at the same level of abstraction of the experts. The idea is that to produce an efficient diagnosis program, these models must use the same assumptions the experts use to reason about the behaviour of the dynamic system to diagnose. The main advantages of this approach are the following: firstly, the resulting models are compatible with Reiter's theory of diagnosis and secondly, the approach is specifically designed to be applied to dynamic processes.

The paper presents an application of this approach to the Cublize dam degradation, with the only expertise of a chapter of Peyras PHD manuscript, an expert of the Cemagref. The resulting models have been validated by the Cemagref experts.

Our current works aim at formalizing the global methodology and to design a diagnosis algorithm based on the use of the timed observations contained in the behavioural model to limit the hypothesis.

Acknowledgments

Thanks to Corinne Curt (Cemagref) for her assistance and our discussions.

References

Basseville, M., and Cordier, M.-O. 1996. Surveillance et diagnostic de systèmes dynamiques : approches complémentaires du traitement de signal et de l'intelligence artificielle. *Office Publication*.

Bouché, P.; Le Goc, M.; and Giambiasi, N. 2005. Modeling Discrete Event Sequences for Discovering Diagnosis Signatures. *Summer Computer Simulation Conference, SCSC'05, Philadelphia, USA.*

Brusoni V.; Console, L.; Terenziani, P.; and Theseider Dupré, D. 1998. A spectrum of definition for temporal model based diagnostic. *Artificial Intelligence, vol. 102, 1:39-79.*

Clancey, W. 1985. Heuristic Classification. *Artificial Intelligence Journal, Vol 25, 3:289-350.*

Cordier M.O., and Dousson, C. 2000. Alarm driven monitoring based on chronicles. Proceedings of SafeProcess 2000, Budapest, Hungary, pp. 286-291.

Chittaro L.; Guida, G.; Tasso, C.; and Toppano, E. 1993. Functional and Teleological Knowledge in the Multimodeling Approach for Reasoning about Physical Systems: a Case Study in Diagnosis. *IEEE transactions on systems*.

Le Moigne, J.-L. 1999. La modélisation des systèmes complexes. *Dunod.*

Le Goc, M. 2004. SACHEM, a real Time Intelligent Diagnosis System based on the Discrete Event Paradigm. *Simulation, The Society for Modeling and Simulation International Ed., Vol. 80, 11:591-617.*

Masse, E. ; Curt, C. and Le Goc, M. 2007. Développement d'une méthode pour la maîtrise de la sécurité des barrages. To appear in the Proceedings of the 25ième Rencontres Universitaires de Génie Civil, Bordeaux, France, 23-25 mai.

Peyras, L. 2003. Diagnostic et analyse de risques liés au vieillissement des barrages, Développement de méthodes d'aide à l'expertise. PHD, Université Blaise Pascal, Clermont II.

Reiter, R. 1987. A theory for Diagnosis from First Principles. Artificial Intelligence, 32, pp. 57-95.

Schreiber, A.; Th.; Akkermans; J. M.; Anjewierden, A. A.; de Hoog, R.; Shadbolt, N. R.; Van de Velde, W.; and Wielinga, B. J. 2000. Knowledge Engineering and Management, The CommonKADS methodology », MIT Press.

Thetiot, R. 1999. Utilisation de l'approche multi-modèles pour l'aide au diagnostic d'installations industrielles. PHD, Université d'Evry Val d' Essonne.

Traoré, M. 2003. A Meta-Theoretic Approach to Modeling and Simulation. In Proceedings of the 2003 Winter Simulation Conference, S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, eds, pp. 604-610.

Zanni, C.; Le Goc, M.; and Frydman, C. 2006. A Conceptual Framework for the Analysis, Classification and Choice of Knowledge-Based Diagnosis Systems. International Journal of Knowledge-Based & Intelligent Engineering Systems (KES Journal), IOS Press Eds., Vol. 10, 2:113-138.

Zouaoui, F. 1998. Aide à l'interprétation du fonctionnement des systèmes phy-siques en utilisant une approche multi-modèles. Application au circuit primaire d'une centrale à eau pressurisée », Université de Paris XI.

Designing Resource-Bounded Reasoners using Bayesian Networks: System Health Monitoring and Diagnosis

Ole J. Mengshoel
RIACS
NASA Ames Research Center
Mail Stop 269-3
Moffett Field, CA 94035
omengshoel@riacs.edu

Abstract

In this work we are concerned with the conceptual design of large-scale diagnostic and health management systems that use Bayesian networks. While they are potentially powerful, improperly designed Bayesian networks can result in too high memory requirements or too long inference times, to they point where they may not be acceptable for real-time diagnosis and health management in resource-bounded systems such as NASA's aerospace vehicles. We investigate the clique tree clustering approach to Bayesian network inference, where increasing the size and connectivity of a Bayesian network typically also increases clique tree size. This paper combines techniques for analytically characterizing clique tree growth with bounds on clique tree size imposed by resource constraints, thereby aiding the design and optimization of large-scale Bayesian networks in resource-bounded systems. We provide both theoretical and experimental results, and illustrate our approach using a NASA case study.

Introduction

In this work we take a systems point of view, and consider situations where a diagnostic process needs to be carefully designed within an overall computer system architecture. Such computational considerations come to the forefront in diagnosis applications where resources are severely limited. For example, diagnosis plays or could play a key role in many real-time systems with limited memory resources.

While we assume a Bayesian network approach to diagnosis, our work carries over to other model-based approaches. Bayesian networks provide a well-established approach to model-based diagnosis and monitoring (Andreassen *et al.* 1987; Shwe *et al.* 1991; Lerner *et al.* 2000; Roychoudhury, Biswas, & Koutsoukos 2006), and clique tree clustering is an important Bayesian network inference algorithm (Lauritzen & Spiegelhalter 1988; Andersen *et al.* 1989). Using the tree clustering approach, a BN's directed graph is compiled to an undirected graph, a clique tree (Lauritzen & Spiegelhalter 1988). In order to compute marginals or most probable explanations, evidence is propagated in the clique tree. The performance of clique tree clustering algorithms depends on the treewidth or the optimal maximal clique size of a BN's induced clique tree (Dechter & Fattah 2001). Unfortunately, it turns out that maximal clique size and total clique tree size can be prohibitively large in application BNs (Shwe *et al.* 1991) as well as in relatively small synthetic BNs (Mengshoel, Wilkins, & Roth 2006).

Early in system design, during conceptual design, there is a lack of information regarding likely BN topologies and consequently much is not known with respect to BN inference times as well as accuracy of diagnostic inference. Ideally, a BN designer would like to carefully investigate important but difficult questions such as the following:

- What is the *resource consumption*, with respect to time and space, of different BN models for diagnosis or system health management?
- What are the *resource bounds* imposed by different hardware and software platform alternatives?

We address these questions by using and extending high-level, macroscopic models of clique tree and inference time growth (Mengshoel, Wilkins, & Roth 2006; Mengshoel 2007). Our growth curves are based on theoretical considerations and on empirical data created through the use of problem instance generators (Mitchell, Selman, & Levesque 1992; Mengshoel, Wilkins, & Roth 2006; Mengshoel 2007). In this paper we show how clique tree growth curves, along with resource bounds, can be used for trade studies during conceptual design. NASA's next-generation crew launch vehicle is used as a case study (NASA 2005).

We believe this research is significant because resource constraints are fundamental in computer science and engineering, including in diagnostic and health management algorithms and software. This work enables improvements in how resource-bounded systems, in particular real-time systems with embedded health monitoring components, are designed and analyzed.

The remainder of this paper is organized as follows. First, we introduce a few fundamental concepts and results related to Bayesian networks. We then briefly discuss the resource-bounded nature of system health management applications of interest to NASA. Next, we present a framework for analyzing resource bounds and resource consumption, the latter as a function of parameters that characterize Bayesian networks. Finally, we apply our analytical framework in computational experiments.

Preliminaries

A Bayesian network (BN) is a tuple $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$, where $(\boldsymbol{X}, \boldsymbol{E})$ is a DAG with associated conditional probability distributions $\boldsymbol{P} = \{\Pr(X_1 \mid \Pi_{X_1}), \ldots, \Pr(X_n \mid \Pi_{X_n})\}$. Here, $\Pr(X \mid \Pi_X)$ is the conditional probability distribution (CPT) for $X \in \boldsymbol{X}$ given its parents Π_X. Let π_X represent an instantiation of Π_X. The independence assumptions encoded in $(\boldsymbol{X}, \boldsymbol{E})$ imply the joint probability distribution

$$\Pr(\boldsymbol{x}) = \prod_{i=1}^{n} \Pr(x_i \mid \pi_{X_i}), \quad (1)$$

where $\Pr(\boldsymbol{x}) = \Pr(X_1 = x_1, \ldots, X_n = x_n)$.

There are two broad classes of approaches to Bayesian network inference: Interpretation and compilation. In interpretation approaches, a Bayesian network is directly used for inference. In compilation approaches, a Bayesian network is off-line compiled into a secondary data structure, where the details depend on the approach being used, and this secondary data structure is then used for on-line inference. Due to their high level of predictability and fast execution times, compilation approaches are especially suitable for resource-bounded reasoning and real-time systems. Our focus here is therefore on compilation approaches, and in particular the arithmetic circuit approach and the clique tree clustering (or join tree) approach.

Creation of an arithmetic circuit from a Bayesian network is a relatively recent compilation approach (Park & Darwiche 2004; Chavira & Darwiche 2007). Here, Bayesian networks are represented as multivariate polynomials, translated into arithmetic circuits, and thus probabilistic inference translates into a process of operating on such circuits. These circuits have a relatively simple structure, but can be used to answer a wide range of probabilistic queries.

In this paper, though, our main focus is on the clique tree clustering approach (Lauritzen & Spiegelhalter 1988; Andersen *et al.* 1989). A clique tree β''' is constructed from a BN in the following way by the Hugin tree clustering algorithm (Andersen *et al.* 1989). First, an initial moral graph β' is constructed by making an undirected copy of β and then augmenting it as follows. For each node X in β, Hugin adds to β' an edge between each pair of nodes in Π_X if no such edge already exists in β'. Second, Hugin creates a triangulated graph β'' by adding fill-in edges to β' such that no chordless cycle of length greater than three exists. Third, a clique tree β''' is created from β''. Hugin uses essentially the same clique tree β''' for both belief updating (marginals) and belief revision (most probable explanations).

There are several reasons why we investigate tree clustering here. First, tree clustering is a theoretically well-founded approach to exact inference in BNs. The complexity of other exact BN inference algorithms — including conditioning and elimination algorithms — depends on treewidth τ^*, which is closely related to tree clustering's optimal maximal clique size h^* since $\tau^* = h^* - 1$ (Lauritzen & Spiegelhalter 1988; Dechter & Fattah 2001). Second, due to the use of clique trees for inference, tree clustering is quite predictable, which is a major advantage in resource-bounded environments. Third, tree clustering algorithms have been implemented in several high-quality software systems.

We investigate bipartite BNs, BNs in which the nodes $\boldsymbol{X}$ are partitioned into root nodes $\boldsymbol{V}$ and leaf nodes $\boldsymbol{C}$. Bipartite BNs are sampled using the BPART algorithm (Mengshoel, Wilkins, & Roth 2006), which is a generalization of an algorithm for randomly generating instances of the satisfiability problem (Mitchell, Selman, & Levesque 1992). The BPART algorithm operates as follows. First, $V = |\boldsymbol{V}|$ root nodes, each with S states, are created. Second, $C = |\boldsymbol{C}|$ leaf nodes, also each with S states, are created. For each leaf node, P parent nodes $\{X_1, \ldots, X_P\}$ are picked uniformly at random without replacement among the V root nodes. In this paper our main interest is in the structural parameters V, C, P, and S, and we do not consider the CPTs constructed by BPART. Consequently, the signature BPART(V, C, P, S) is used. The number of BN edges E created is given by $E = C \times P$ and the total number of nodes is $N = C + V$.

Our key idea is that growth curves provide estimates of resource consumption in terms of clique tree size. Such clique trees do not need to be generated from bipartite BNs, even though we emphasize them here. Bipartite BNs are important in applications. Naive Bayes models, successful in spam filtering, are a special case of bipartite BNs with only one root node. The QMR-DT BN is a bipartite medical BN — diseases are root nodes and symptoms are leaf nodes (Shwe *et al.* 1991). Special inference algorithms have also been designed for bipartite BNs (Ng & Jordan 2000).

For all BNs, including for bipartite BNs, it is important but also very difficult to understand and predict clique tree clustering's cycle-generation and fill-in processes. Further, these strongly non-linear processes need to be traded off against resource bounds. We will return to these topics after first introducing our perspective on resource-bounded diagnostic reasoning.

Designing Resource-Bounded Reasoners

Diagnosis and system health management in resource-bounded systems is of great interest to NASA and the aerospace community. Relevant examples of Bayesian networks exist, for example, in process monitoring and control (Lerner *et al.* 2000; Roychoudhury, Biswas, & Koutsoukos 2006) and medical diagnosis and monitoring (Andreassen *et al.* 1987; Shwe *et al.* 1991).

As a concrete case study, we will discuss NASA's next-generation crew launch vehicle (CLV). This vehicle, which will replace the space shuttle, is currently being developed. The CLV is likely to include a solid rocket engine; see Figure 1. Historically, a 91.4% success rate was observed for world-wide space launches in the time period 1957-2004, with the propulsion subsystem being the Achilles' heel of launch vehicles (Chang & Tomei 2005). To ensure the safety of the crew, it is important to estimate and monitor the health state of the solid rocket during launch, such that the crew can escape in a timely fashion if needed.

In a study of space exploration system architectures, NASA identified (i) key solid rocket failure modes, (ii) their potential detection methods, and (iii) approximate reaction times for crew abort (NASA 2005). Failure modes include

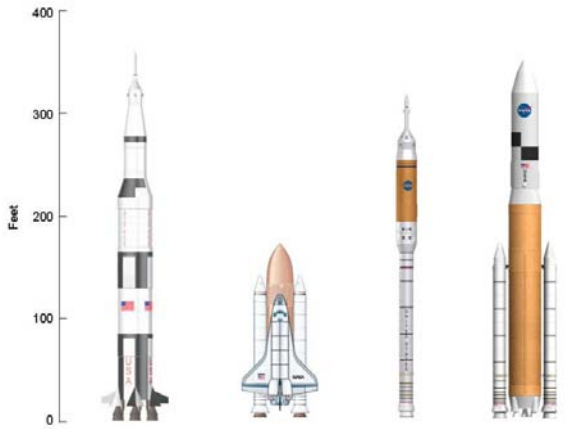


Figure 1: From left to right: The Saturn V, the Space Shuttle and two candidate designs for the next-generation Crew Launch Vehicle (CLV).

joint failures, structural failures, thermal failures, and performance failures. A total of 22 key failure modes have been identified (NASA 2005), however as the system is being developed this number might change. An interesting question, which we will return to in the experimental section, is what the impact would be if the number of failure modes or sensors were increased or decreased. Possible sensors include traditional cameras, infrared cameras, thermocouples, pressure transducers, burn-through wires, and strain gauges. Reaction times, which are hard real-time, range from close to 0 seconds for catastrophic failures such as case failure to 130 seconds for less threatening failures such as case insulator failures (NASA 2005).

Musliner and his coauthors identified three approaches to real-time AI (Musliner *et al.* 1995); we investigate what they call "embedding AI into a real-time system". In other words, we make system health management (SHM) a real-time task or a set of a real-time tasks. We call this embedded system health management (ESHM).

The ESHM approach is based on real-time tasks and may also utilize a hard real-time operating system (RTOS). A prototypical RTOS model has emerged over the last decade or so: An RTOS typically uses priority-based preemptive scheduling where each task has a fixed priority. The RTOS scheduler lets higher-priority tasks interrupt lower-priority tasks while tasks on the same priority level, if supported, are executed on a round-robin basis. A periodic RTOS task $\tau_i = (p_i, d_i, w_i)$ has a task period p_i, a deadline d_i, and a worst-case execution time (WCET) w_i. In other words, a task not only needs to compute correct outputs for all inputs, it also needs to do so within its deadline d_i. Formally, let T_i be the random execution time for τ_i; clearly $\Pr(T_i > w_i) = 0$ needs to hold. Here, T_i accounts for the different execution paths of τ_i due to input variations. Different RTOS scheduling algorithms exist. Two approaches that are amendable to mathematical analysis are known as earliest deadline first (EDF) scheduling and rate-monotonic (RM) scheduling. Through analysis, and given appropriate assumptions, one can determine whether all task deadlines can be met for a given set of tasks under RM or EDF.

Clearly, some AI approaches are suitable for this ESHM approach, while others are not (Musliner *et al.* 1995). Among the real-time task parameters discussed above, a predictable, bounded worst-case execution time w_i is essential. Clique tree size, along with the application-dependent hardware and software platforms, essentially determine w_i. Further, once a clique tree has been generated by compilation, its memory requirement has been determined, which is also desirable in real-time and resource-bounded systems. All in all, BN inference using tree clustering provides a solid foundation for resource-bounded reasoning.

In the conceptual design phase, there is typically a lack of detailed knowledge about the overall system being designed, with the implication that a large number of candidate ESHM BNs exists. Generation of all these BNs and processing them using clique tree clustering is seldom feasible. At the same time, given the computational hardness of BN inference (Cooper 1990; Shimony 1994) as well as the observed difficulty of application BNs (Andreassen *et al.* 1987; Shwe *et al.* 1991; Lerner *et al.* 2000; Roychoudhury, Biswas, & Koutsoukos 2006) and synthetic BNs (Mengshoel, Wilkins, & Roth 2006) it is clear that a certain amount of care is well-advised, in particular when the size and connectivity of Bayesian networks increase. This motivates our discussion of resource bounds and resource consumption in the following.

Resource Bounds

Memory and time bounds are fundamental in computer science, including in SHM algorithms and software. In this section we develop novel bounds on clique tree size to express time- and memory-bounds.

Clique tree sizes can be translated into quantities directly indicating memory usage, once a memory model such as the following is introduced.

Definition 1 (Linear memory model) *Let $\phi = \beta'''$ be a clique tree with size $k(\phi)$. A linear memory resource model for ϕ is given by*

$$r_m(\phi) = a \times k(\phi) \text{ bytes} + b \text{ bytes},$$

where $a \in \mathbb{N}^+$ and $b \in \mathbb{N}$ are system parameters.

The parameters a and b that determine the amount of primary memory (RAM) required for processing are system-dependent. Here is an example.

Example 2 *Consider a clique tree ϕ, and suppose that each clique tree component uses a* `double` *data type requiring $a = 8$ bytes while the fixed overhead is $b = 0$ bytes. The amount of RAM r_m needed to accommodate a clique tree consisting of one clique with $h = 24$ binary ($S = 2$) nodes is then:*

$$r_m(\phi) = a \times k(\phi) + b \text{ bytes} = 8 \times S^h \text{ bytes} = 2^{24} \times 8 \text{ bytes} = 134,217,728 \text{ bytes} = 128MB.$$

We now introduce memory-induced clique tree bounds $\lambda_m(m_{\max})$ as follows.

Definition 3 (Memory-bounded maximal clique tree) *Let Φ be a set of clique trees, and put $\Phi' = \{\phi \mid \phi \in \Phi$ and $r_m(\phi) \leq m_{\max}\}$. The memory-bounded maximal clique tree size $\lambda_m(m_{\max})$, determined by Φ and the memory bound $m_{\max}$, is defined as*[1]

$$\lambda_m(m_{\max}) = k\left(\arg\max_{\phi\in\Phi'} r_m(\phi)\right). \quad (2)$$

In words, Definition 3 expresses the idea of picking as large a clique tree as possible without going beyond the memory bound $m_{\max}$. The sizes of the diagnostic BNs we can develop are constrained by the maximal memory $m_{\max}$ available to their clique trees.

To be useful for real-time diagnosis and health management, not only must an ESHM Bayesian network fit into memory, it must also support the timely computation of a diagnosis. To capture this constraint, we introduce the following linear time model.

Definition 4 (Linear time model) *Let ϕ be a clique tree. A linear WCET time model for ϕ is given by*

$$r_t(\phi) = c \times r_m(\phi)\ ms + d\ ms,$$

where $c \in \mathbb{N}^+$ and $d \in \mathbb{N}$ are system parameters.

Definition 4 expresses the fact that execution time $r_t(\phi)$ is partly determined by clique tree memory requirements $r_m(\phi)$, but $r_t(\phi)$ also depends on several hardware- and software-specific factors as reflected in the parameters c and d. Hardware-specific factors include the processor clock speed, amount of primary memory, speed of primary memory, and amount of cache. Software-specific factors include the type of data structures, programming language, compiler, and operating system used to implement and deploy the clique tree clustering system. We leave detailed investigations of the impact of different hardware and systems software for future research.

We now introduce the time-induced clique tree bound $\lambda_t(t_{\max})$ as follows. For simplicity, we consider exactly one ESHM RTOS task $\tau_1 = (p_1, d_1, w_1)$, and put $t_{\max} = w_1$.

Definition 5 (Time-bounded maximal clique tree) *Let Φ be a set of clique trees, and put $\Phi' = \{\phi \mid \phi \in \Phi$ and $r_t(\phi) \leq t_{\max}\}$. The time-bounded maximal clique tree size $\lambda_t(t_{\max})$, determined by Φ and the time bound $t_{\max}$, is defined as*

$$\lambda_t(t_{\max}) = k\left(\arg\max_{\phi\in\Phi'} r_t(\phi)\right). \quad (3)$$

In words, Definition 5 defines the size of the largest clique tree whose execution time does not exceed $t_{\max}$.

During conceptual design, $t_{\max}$ and $m_{\max}$ may in fact not be fixed. The reason for this is that different system and ESHM design alternatives are in general considered early in development, which translates into a range of possible values for $t_{\max}$ and $m_{\max}$. Consequently, a diagnostic system designer may consider subsets $\{m_1, \ldots, m_k\}$ and $\{t_1, \ldots, t_n\}$ rather than $t_{\max}$ and $m_{\max}$ respectively. Let $m \in \{m_1, \ldots, m_k\}$ and $t \in \{t_1, \ldots, t_n\}$. Then we have from (2) and (3) the following joint resource-induced clique tree bound $\lambda(m, t)$:

$$\lambda(m, t) = \min(\lambda_m(m), \lambda_t(t)). \quad (4)$$

Inputting $(m, t) \in \{m_1, \ldots, m_k\} \times \{t_1, \ldots, t_n\}$ into (4) results in a set of resource-induced clique tree bounds $\Lambda := \{\lambda_1, \lambda_2, \ldots\}$. The advantage of using resource-induced bounds Λ on total clique tree size (or, along similar lines, on arithmetic circuit size) is that they uniformly represent both memory bounds and time bounds, and can easily be compared to clique tree growth curves expressing resource consumption. In the experimental section, we will use a set $\{\lambda_1, \lambda_2, \lambda_3\}$ of resource-induced bounds.

Resource Consumption

As the size and connectivity of a BN increases, so does the consumption of computational resources required for its processing. Therefore, when embarking on a BN development effort, one would like to avoid developing a BN model that cannot be processed within the resource bounds of a particular computer system. To avoid this problem, we estimate resource consumption ahead of time by analytical means. Here, we discuss an analytical framework for characterizing clique tree growth (Mengshoel 2007).

In a diagnostic bipartite BNs, the number of root nodes V represents the number of failure modes, components, or diseases, and the number of leaf nodes C represents the number of sensors, tests, or symptoms. From a bipartite BN, compilation produces a clique tree consisting of root cliques and mixed cliques (Mengshoel, Wilkins, & Roth 2006). *Root cliques* are cliques with root nodes only, while *mixed cliques* are cliques with both root nodes and leaf nodes.

Based on previous research (Mengshoel, Wilkins, & Roth 2006; Mengshoel 2007), we now provide a qualitative discussion of the growth of clique trees generated from BPART BNs in terms of the C/V-ratio. We identify three broad stages of clique tree growth: The initial growth stage, the rapid growth stage, and the saturated growth stage. The *initial growth stage*, where the C/V-ratio is "low" (for $P = 2$, up to approximately $C/V \approx 1$), is characterized by "few" leaf nodes relative to the number of root nodes. There is consequently a relatively low contribution by root cliques to the clique tree. This stage is generally dominated by mixed cliques — indeed as $C/V \to 0$ there are no root cliques with more than one root node. During the *rapid growth stage*, where the C/V-ratio is "medium" (for $P = 2$, from approximately $C/V \approx 1$), non-trivial root cliques start appearing, causing dramatic growth in the total size of root cliques on average. This growth is due to formation of cycles where fill-in edges are required in order to triangulate the moral graph. Because of fill-in edges, the root cliques gradually dominate the mixed cliques in terms of their contribution to total clique tree size. The *saturated growth stage*, where the C/V-ratio is "high", is characterized by a "large" number of leaf nodes relative to the number of root nodes. As $C/V \to \infty$, one root clique with V BN root nodes and size

[1] If there are multiple candidates for $\arg\max_{\phi\in\Phi} r_m(\phi)$, we arbitrarily pick one of them since their size is the same.

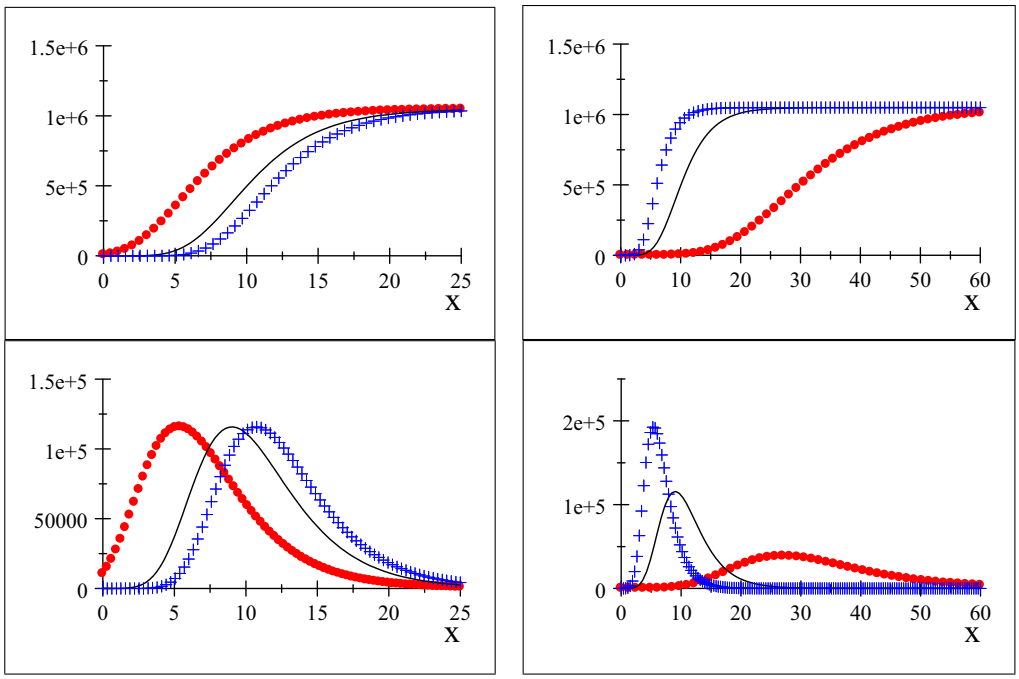


Figure 2: Gompertz curves of the form $g(x) = g(\infty)e^{-\zeta e^{-\gamma x}} = 2^{20}e^{-\zeta e^{-\gamma x}}$, where the parameters ζ and γ are varied. *Top left:* The curves plotted have the form $g(x) = 2^{20}e^{-\zeta e^{-0.3x}}$, with $\zeta = 5$ (red dotted curve), $\zeta = 15$ (black solid curve), and $\zeta = 25$ (blue crossed curve). *Top right:* The curves plotted have the form $g(x) = 2^{20}e^{-15e^{-\gamma x}}$, with $\gamma = 0.1$ (red dotted curve), $\gamma = 0.3$ (black solid curve), and $\gamma = 0.5$ (blue crossed curve). *Bottom left:* Growth rates $g'(x)$ for different $g(x)$ directly above. *Bottom right:* Growth rates $g'(x)$ for different $g(x)$ directly above.

S^V emerges. The phase of greatest interest to developers of large-scale BNs is the rapid growth stage, where the size of induced clique trees quickly transition from feasible to infeasible for resource-bounded reasoners.

Clique trees are discrete structures, however we here use continuous growth models in order to simplify mathematical analysis.

Definition 6 (Clique tree growth curve) *Let $g_R(x)$ be the growth curve for all root cliques and $g_M(x)$ the growth curve for all mixed cliques. The (total) clique tree growth curve is defined as*

$$g_T(x) = g_R(x) + g_M(x).$$

Restricted growth has been modeled using different sigmoidal growth curves ("S-curves"), including the logistic, Gompertz, Complementary Gompertz, and Richards growth curves (Banks 1994; Lindsey 2004). We emphasize Gompertz growth curves, since they are theoretically plausible and turn out to give best fit empirically (Mengshoel 2007).

Definition 7 (Gompertz growth curve) *The Gompertz growth curve is*

$$g(x) = g(\infty)e^{-\zeta e^{-\gamma x}}, \quad (5)$$

where $g(\infty)$ is the asymptote as $x \to \infty$.

The independent variable x in (5) is $x = C$ or $x = C/V$ in this paper. The derivative

$$g'(x) = \frac{d}{dx}\left(g(\infty)e^{-\zeta e^{-\gamma x}}\right) = g(\infty)\zeta\gamma e^{-\gamma x}e^{-\zeta e^{-\gamma x}},$$

which we call the growth rate, shows how Gompertz growth changes as a function of x.

We now introduce, from related work (Mengshoel 2007), a Gompertz growth curve for BPART.

Theorem 8 (Total Gompertz growth curve) *The total Gompertz growth curve for clique trees generated from BPART(V, C, P, S) BNs, where $x = C$ is the independent parameter, is*

$$g_T(x) = S^V e^{-\zeta e^{-\gamma x}} + xS^{P+1}. \quad (6)$$

Growth curves are frequently used in medicine, biology, and sociology (Banks 1994; Lindsey 2004), however our use of them in designing resource-bounded reasoning systems is, to our knowledge, novel.

In Figure 2 we investigate how varying the parameters ζ and γ impacts the shape of Gompertz curves. The factor $g_R(\infty) = S^V = 2^{20}$ is obtained by considering bipartite Bayesian networks with $V = 20$ binary (so $S = 2$) root nodes. Figure 2 also shows how the growth rate $g'(x)$ changes when ζ and γ are varied.

Let us first consider varying ζ as illustrated to the left in Figure 2. By increasing ζ, the x-location of maximal growth

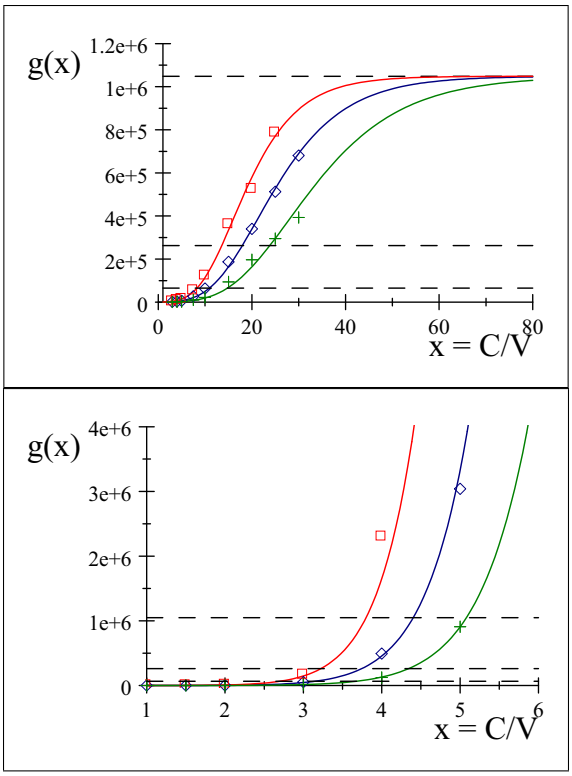


Figure 3: Gompertz growth curves g_{20} (top, for $V = 20$ root nodes) and g_{40} (bottom, for $V = 40$ root nodes) are shown here, along with sample means, using $x = C/V$ as the independent variable. Data from which the curves were created is shown in Table 1. Straight lines represent upper bounds on clique tree size and reflect different resource bounds.

rate $g'(x)$ is increased as well. In other words, the *initial growth stage* lasts longer as ζ increases, and the onset of the *rapid* and *saturated growth stage* are delayed. However, the maximal value of $g'(x)$ does not change with changing ζ.

Let us next consider varying γ as illustrated to the right in Figure 2. As γ increases, the x-location of maximal growth rate $g'(x)$ decreases. For example, $\gamma = 0.1$ has maximal $g'(x)$ at approximately 28, while $\gamma = 0.5$ has maximal $g'(x)$ at approximately 6. In addition, with increasing γ the maximal value of $g'(x)$ increases. In other words, the *initial growth stage* is shorter as γ increases, and the *rapid* and *saturated growth stage* start for smaller values of x.

It is clear from Figure 2 that Gompertz curves provide modelling flexibility and power that is potentially useful in modelling growth in resource consumption of model-based techniques including BNs. For instance, it can be very useful to know where a BN or a distribution of BNs is located on $g(x)$ and $g'(x)$.

Experimental Results

As a concrete case study, we investigate the design of an ESHM diagnostic system for a next-generation crew launch vehicle (CLV) (NASA 2005). The identified failure modes and detection methods can be represented as a bipartite Bayesian network where the failure modes are root nodes, and detectors or sensor are leaf nodes. Such a bipartite network could make up a time-slice in a dynamic, time-sliced Bayesian network, where each time slice contains both discrete and continuous random variables. Our goal is to obtain high diagnostic accuracy while also keeping clique tree size (and therefore memory requirement and inference time) below certain bounds. To achieve this, we consider upper and lower bounds on the number of root and leaf nodes. Formally, we introduce the parameters $V_{\min}$, $V_{\max}$, $C_{\min}$, and $C_{\max}$. $V_{\min}$ represents the minimal number of root nodes considered for the BN; $V_{\max}$ is the maximal number of root nodes. Similarly, $C_{\min}$ and $C_{\max}$ are bounds for the leaf nodes in a bipartite BN.

Empirical Gompertz Growth Curves
$g^-_{20}(x) = 2^{20}\exp(-\exp(2.158)\exp(-0.0769x))$
$g_{20}(x) = 2^{20}\exp(-\exp(2.108)\exp(-0.0993x))$
$g^+_{20}(x) = 2^{20}\exp(-\exp(2.113)\exp(-0.132x))$
$g^-_{40}(x) = 2^{40}\exp(-\exp(3.288)\exp(-0.130x))$
$g_{40}(x) = 2^{40}\exp(-\exp(3.269)\exp(-0.145x))$
$g^+_{40}(x) = 2^{40}\exp(-\exp(3.259)\exp(-0.166x))$

Table 2: Growth curves constructed from bipartite BNs with $V = 20$ and $V = 40$ root nodes. The underlying data is summarized in Table 1.

In experiments reflecting the current estimate $V = 22$ CLV failure modes (NASA 2005), we sampled bipartite BNs using BPART(V, C, 2, 2), using $V = V_{\min} = 20$ and $V = V_{\max} = 40$. The results reported here are based on experiments with a total of 2,700 BNs, with 1,800 BNs for $V_{\min} = 20$ and 900 BNs for $V_{\max} = 40$.

Results are reported in Table 1 and Figure 3. Along the figure's x-axis, we use $x = C/V$ as the independent parameter. Along the y-axis, we display the empirically determined Gompertz growth curves for the root cliques:

$$g(x) = g_R(x) = S^V e^{-\zeta e^{-\gamma x}}, \qquad (7)$$

as well as results from the samples. We focus on the root clique growth curve $g_R(x)$ since it dwarfs the mixed clique growth curve $g_M(x)$ for the parameter values of interest here. In (7), we determined ζ and γ by experimentation. These two parameters therefore depend on C, V, P, and S as well as the BNs sampled. To determine a given pair of ζ and γ, an estimation approach based on Lindsey's (Lindsey 2004) was employed — see Figure 4 and elsewhere (Mengshoel 2007) for details.

For each value of V we present three growth curves, namely for minimums $g^-_V(x)$, for means $g_V(x)$, and for maximums $g^+_V(x)$. Maximums are conservative estimates and reflect, for example, situations in which minimal effort is available for optimization of the BN's structure in order to reduce clique tree size.

Figure 3 contains the following curves:

- The growth curves $g^-_{20}(x)$, $g_{20}(x)$, and $g^+_{20}(x)$ represent

C/V	1	1.5	2	3	4	5	7.5	10	15	20	25
Min, $V = 20$				584	1264	2312	10304	22016	94208	196608	294912
Mean, $V = 20$				1204	3221	6683	27292	63245	187580	339838	512953
Max, $V = 20$				2568	9712	13888	53504	122880	360448	524288	786432
Min, $V = 40$	54	392	996	15552	123696	907728					
Mean, $V = 40$	150	658	3010	52988	492283	3040530					
Max, $V = 40$	274	1232	9754	165000	2299072	9869824					

Table 1: Raw data and sample means for total size of root cliques in cliques trees, generated by varying the C/V-ratio for $V = 20$ and $V = 40$ root nodes. This data was used to create the Gompertz growth curves in Figure 3.

clique tree sizes for the minimal number of failure modes (or causes) $V_{\min} = 20$.

- The growth curves $g^-_{40}(x)$, $g_{40}(x)$, and $g^+_{40}(x)$ represent clique tree sizes for the maximal number of failure modes $V_{\max} = 40$.
- A straight line $\lambda_i \in \Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ represents a resource-induced clique tree bound, where λ_i is either a time-induced bound or a memory-induced bound according to (4). Here, $\lambda_1 = 2^{16}$, $\lambda_1 = 2^{18}$, and $\lambda_1 = 2^{20}$.

Table 2 contains empirically determined formulas for the growth curves. One can consult these curves to determine which point(s) or area(s) represents a reasonable trade-off between the various factors. We now give a few examples.

Suppose that a diagnostic system designer wants to investigate the impact of varying the size of the sensor suite for a CLV. If the number of sensors C is increased, then C/V increases as well. It is thus interesting to consider the different impact of large C/V values for $V = 20$ versus $V = 40$. For sake of argument, suppose that $x = C/V \geq 4$; i.e. there are four or more times as many sensors as state variables in a BN and we study the impact of varying $\lambda \in \Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$. Consulting Figure 3, we easily see that for $V = 20$, $C/V \approx 4$ is not a problem, even for $\lambda_1 = 2^{16}$. For $V = 40$, on the other hand, λ_1 is clearly a challenge, even for the most optimistic curve $g^-_{40}(x)$. On the other hand, $g_{40}(4) < \lambda_3$ and thus λ_3 gives a little bit of a margin, while $g_{40}(4) > \lambda_2 > g^-_{40}(4)$. Using λ_2 thus means taking some risk, and one potentially needs to spend some time optimizing the BN or the inference algorithm.

We now discuss the growth curve formulas in Table 2. The biggest difference here, for both $V = 20$ and $V = 40$, is perhaps in the γ parameter. This shows not only an earlier maximal growth rate, as reflected in Figure 2, but also greater maximal growth rate for $g^+_V(x)$ compared to those for $g^-_V(x)$ and $g_V(x)$. In other words, if we are concerned about conservative estimates of clique tree growth as reflected in $g^+_V(x)$, and are in or approaching the rapid growth stage, then this results suggests that clique tree size can increase very dramatically as a result of rather minor increases in C. This effect is, we believe, relatively easy to see if one compares the growth curves in Table 2, but rather difficult to catch if one only looks at the data in Table 1.

Conclusion and Future Work

The goal of this paper has been to help bridge the gap between, on the one hand, diagnostic resource-bounded reasoning and, on the other hand, Bayesian network inference using clique tree clustering. Our conclusion is two-fold. First, diagnostic reasoning in resource-bounded and real-time systems is as much about predictability as speed. And clique tree clustering, being a dynamic programming algorithm, is in fact quite predictable once a particular BN has been developed and compiled into a static clique tree. A similar argument holds for the arithmetic circuit approach.

Second, we have discussed the use of growth curves in trade-off studies during conceptual design, where only information about distributions of BNs is available. Conceptual design presents a chicken-and-egg problem, since in order to compute clique tree size or inference time one needs to construct one or more Bayesian networks. However, in order to develop a Bayesian network by hand, one needs to perform extensive knowledge engineering, which is beyond the scope of conceptual design, especially for large-scale diagnostic BNs.

To tackle the challenge of conceptual design of Bayesian networks, we have developed a design paradigm utilizing computational resource bounds along with coarse-grained growth models based on processing sampled BNs. Our approach facilitates a better understanding of the trade-offs between the resource demands of clique trees (or arithmetic circuits) created from different BNs versus computational resource bounds. The approach has the potential to lead to fewer problems due to under-engineered systems during system implementation and integration, thus reducing cost and risk. Since more is known early in design, we potentially also reduce the chance of over-engineered systems, thereby potentially reducing their cost and weight, both major concerns in aerospace as well as in other disciplines.

This research can be extended in several directions, of which we mention a few. First, it would be interesting to consider in more detail BNs beyond the bipartite model and more generally extend the class of BNs characterized using growth curves. Second, it could be useful to study other model-based approaches, algorithms, and applications where resource bounds are relevant.

Acknowledgments This material is based upon work supported by NASA under award NCC2-1426. The anonymous reviewers are acknowledged for their comments, which helped improve the article.

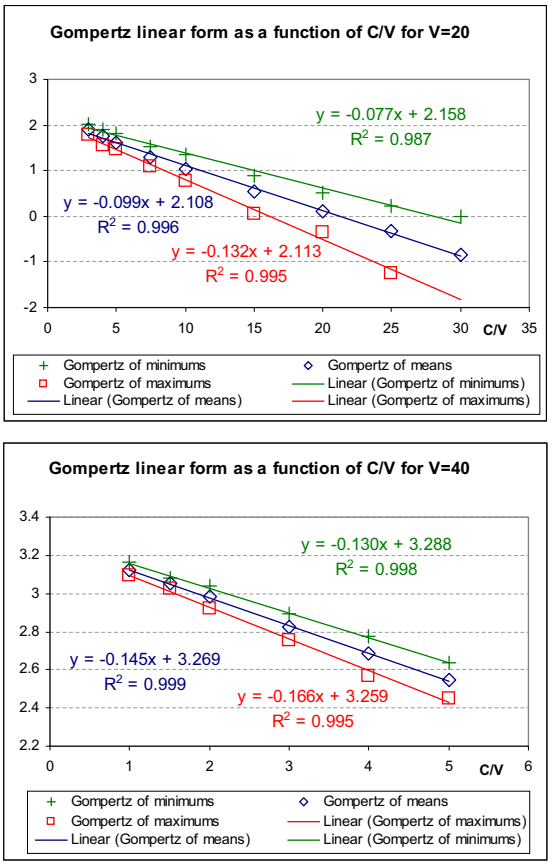


Figure 4: Linear forms summarizing the construction of the Gompertz growth curves are shown here. The regression lines provide the parameters ζ and γ used in the Gompertz growth curves.

References

Andersen, S. K.; Olesen, K. G.; Jensen, F. V.; and Jensen, F. 1989. HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 2, 1080–1085.

Andreassen, S.; Woldbye, M.; Falck, B.; and Andersen, S. 1987. MUNIN – A causal probabilistic network for interpretation of electromyographic findings. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 366–372.

Banks, R. B. 1994. *Growth and Diffusion Phenomena*. New York: Springer.

Chang, I., and Tomei, E. J. 2005. Solid rocket failures in world space launches. In *41st AIAA Joint Propulsion Conference*.

Chavira, M., and Darwiche, A. 2007. Compiling Bayesian networks using variable elimination. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2443–2449.

Cooper, F. G. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42:393–405.

Dechter, R., and Fattah, Y. E. 2001. Topological parameters for time-space tradeoff. *Artificial Intelligence* 125(1-2):93–118.

Lauritzen, S., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B* 50(2):157–224.

Lerner, U.; Parr, R.; Koller, D.; and Biswas, G. 2000. Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of the Seventeenth national Conference on Artificial Intelligence (AAAI-00)*, 531–537.

Lindsey, J. K. 2004. *Statistical Analysis of Stochastic Processes in Time*. Cambridge: Cambridge.

Mengshoel, O. J.; Wilkins, D. C.; and Roth, D. 2006. Controlled generation of hard and easy Bayesian networks: Impact on maximal clique tree in tree clustering. *Artificial Intelligence* 170(16-17):1137–1174.

Mengshoel, O. J. 2007. Macroscopic models of clique tree growth for Bayesian networks. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*.

Mitchell, D.; Selman, B.; and Levesque, H. J. 1992. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 459–465.

Musliner, D.; Hendler, J.; Agrawala, A. K.; Durfee, E.; Strosnider, J. K.; and Paul, C. J. 1995. The challenges of real-time AI. *IEEE Computer* 28:58–66.

NASA. 2005. Exploration systems architecture study. Technical Report NASA-TM-2005-214062, National Aeronautics and Space Administration.

Ng, A. Y., and Jordan, M. I. 2000. Approximate inference algorithms for two-layer Bayesian networks. In *Advances in Neural Information Processing Systems 12 (NIPS-99)*. MIT Press.

Park, J. D., and Darwiche, A. 2004. A differential semantics for jointree algorithms. *Artificial Intelligence* 156(2):197–216.

Roychoudhury, I.; Biswas, G.; and Koutsoukos, X. 2006. A Bayesian approach to efficient diagnosis of incipient faults. In *Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06)*, 243 – 250.

Shimony, E. 1994. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence* 68:399–410.

Shwe, M.; Middleton, B.; Heckerman, D.; Henrion, M.; Horvitz, E.; Lehmann, H.; and Cooper, G. 1991. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: I. The probabilistic model and inference algorithms. *Methods of Information in Medicine* 30(4):241–255.

Sensor Fault Diagnosis using Linear Interval Observers[1]

Jordi Meseguer[*], Vicenç Puig[*], Teresa Escobet[*], Joseba Quevedo[*], Belarmino Pulido[♦]

[*]Automatic Control Department - Campus de Terrassa - Universidad Politécnica de Cataluña (UPC).Terrassa (Spain)
E-mail: jordimeseguer@hotmail.com; {vicenc.puig, teresa.escobet, joseba.quevedo}@upc.edu
[♦]Departamento de Informática - Universidad de Valladolid
belar@infor.uva.es

Abstract

This work is based on an interval model-based fault diagnosis method that improves the integration of fault detection and isolation tasks. The proposed interface between fault detection and fault isolation considers information about the degree of fault signal activation and the occurrence time of fault signals. Moreover, it uses the combination of several fault signature matrices which store the knowledge about the faulty system behaviour. This paper focuses on how to obtain those matrices when an interval observer is considered in the fault detection stage using the fault sensitivity concept. This allows studying the influence of the fault detection module on the fault isolation stage. Finally, the proposed fault diagnosis approach is applied to detect and isolate faults of the Barcelona's urban sewer system limnimeters (level meter sensors).

Introduction

In model-based fault diagnosis (either using FDI or DX methods), fault detection and fault isolation tasks are considered separately. The typical interface between these two modules is through binary codifying the evaluation of each residual or analytical redundancy relation (ARR) and generating a fault signature. These last years, the integration between fault detection and fault isolation tasks in FDI model-based fault diagnosis has been a very active research area (see among others (Combastel et al. 2003), (Pulido et al. 2005) or (Puig et al. 2005)). On the other hand, the fault effect on a residual set depends mainly on the residual generator used in fault detection. Thus, it can be said that the fault isolation result is deeply affected by the fault detection stage. However, this influence is not only caused because the model-based fault detection has inherent problems as the lack of fault indication persistence (Meseguer et al. 2006c) but also, because the residual generator structure determines the knowledge the fault isolation module has about faults. Thus, in order to diagnose accurately, fault detection and fault isolation can not be considered independently. Besides, when observers are used as fault detection mechanism, the result of the fault detection stage is influenced by the observer gain matrix because it determines the residual generator structure (Meseguer et al. 2006b). Therefore, the theoretical fault signature matrices will be also influenced by the observer gain.

This paper continues the work developed in two previous ones. In particular, (Pulido et al. 2005) presented the need to improve the fault isolation module using additional fault signature matrices instead of only using the binary one. (Meseguer et al 2006c) noticed that the fault isolation result was influenced by the structure of the scheme used in the fault detection stage. This paper goes one step further showing how the theoretical fault signature matrices can be obtained using the fault residual sensitivity concept. Moreover, this allows to show the influence of the fault detection module on the fault isolation stage when sensor faults are considered. Finally, the interval observer-based fault diagnosis algorithm will be applied to the limnimeters of Barcelona's urban sewer system to assess the validity of the derived results. It should be noticed that the fault diagnosis methodology presented in this paper is not restricted to sensor faults. However, since the most frequent faults in an urban sewer system are those affecting to the sensors, this is the main focus of the present paper.

Regarding the structure of the paper, in next section, the passive robust fault detection using interval observers is recalled focusing on the integration between this stage and the fault isolation module. Then, a method to obtain the theoretical fault signature matrices is presented showing the influence of the observer gain. Finally, the last section shows how the fault isolation result is obtained and why this result is affected by the observer gain.

Passive robust based fault detection using interval observers

Interval observer expression

Considering that the system to be monitored can be described by a MIMO linear uncertain dynamic model in discrete-time and in state-space form as

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}(\boldsymbol{\theta})\boldsymbol{x}(k) + \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}(\boldsymbol{\theta})\boldsymbol{x}(k) \end{aligned} \quad (1)$$

without considering faults, disturbances and noise and where $\boldsymbol{A}(\boldsymbol{\theta})$, $\boldsymbol{B}(\boldsymbol{\theta})$, $\boldsymbol{C}(\boldsymbol{\theta})$ are the state, the input and the output matrices respectively, $\boldsymbol{u}(k) \in \Re^{nu}$ and $\boldsymbol{y}(k) \in \Re^{ny}$ are the system input and output vectors, respectively. $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is a set

[1] This work has been funded by the grant CICYT DPI2006-11944 of Spanish Ministry of Education and by Research Comission of the Generalitat de Catalunya (group SAC ref. 2005SGR00537).

of interval bounded parameters representing the model uncertainty: $\boldsymbol{\Theta} = \left\{ \boldsymbol{\theta} \in \Re^{n\theta} \middle| \underline{\boldsymbol{\theta}} \le \boldsymbol{\theta} \le \overline{\boldsymbol{\theta}} \right\}$. This type of model is known as an ***interval model***.

Instead of using directly the system model given by (1) to detect faults, the following state observer will be used:

$$\begin{aligned} \hat{\boldsymbol{x}}(k+1) &= (\boldsymbol{A}(\boldsymbol{\theta}) - \boldsymbol{WC}(\boldsymbol{\theta}))\hat{\boldsymbol{x}}(k) + \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u}(k) + \boldsymbol{W}\boldsymbol{y}(k) \\ \hat{\boldsymbol{y}}(k) &= \boldsymbol{C}(\boldsymbol{\theta})\hat{\boldsymbol{x}}(k) \end{aligned} \tag{2}$$

where $\boldsymbol{W}$ is the observer gain, designed to stabilize the matrix $\boldsymbol{A}_o = \boldsymbol{A}(\boldsymbol{\theta}) - \boldsymbol{WC}(\boldsymbol{\theta})$ and to guarantee a desired fault detection performance for all $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. When $\boldsymbol{W}=\boldsymbol{0}$ (i.e. no correction of the modelled behaviour with sensor measurements is introduced), the observer is in fact a ***simulator*** (Chow et al. 1984). On the other hand, when $\boldsymbol{A} = \boldsymbol{WC}$ (i.e. complete correction of the modelled behaviour with sensor measurements), the observer becomes a ***predictor*** (Chow et al. 1984). (Meseguer et al. 2006a) and (Meseguer et al. 2006b) show that when an observer (i.e., partial correction of the modelled behaviour with sensor measurements) is considered, the fault indication is more persistent for a certain value of $\boldsymbol{W}$ than the corresponding one to the predictor. The observer given by Eq. (2) can be expressed in transfer function form using the *q*-transform and considering zero initial conditions as it follows:

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{G}(q^{-1}, \boldsymbol{\theta})\boldsymbol{u}(k) + \boldsymbol{H}(q^{-1}, \boldsymbol{\theta})\boldsymbol{y}(k) \tag{3}$$

where:

$$\begin{aligned} \boldsymbol{G}(q^{-1}, \boldsymbol{\theta}) &= \boldsymbol{C}(\boldsymbol{\theta})(q\boldsymbol{I} - \boldsymbol{A}_o(\boldsymbol{\theta}))^{-1}\boldsymbol{B}(\boldsymbol{\theta}) \\ \boldsymbol{H}(q^{-1}, \boldsymbol{\theta}) &= \boldsymbol{C}(\boldsymbol{\theta})(q\boldsymbol{I} - \boldsymbol{A}_o(\boldsymbol{\theta}))^{-1}\boldsymbol{W} \end{aligned}$$

The effect of the uncertain parameters $\boldsymbol{\theta}$ on the observer temporal response will be bounded using an interval: $[\,\underline{\hat{y}}(k)\,,\,\overline{\hat{y}}(k)\,]$, where:

$$\underline{\hat{y}}_i(k) = \min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}(\hat{y}_i(k,\boldsymbol{\theta}))\,,\ \overline{\hat{y}}_i(k) = \max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}(\hat{y}_i(k,\boldsymbol{\theta})) \tag{4}$$

This interval can be computed using the algorithm presented in (Puig et al. 2003).

Fault detection using an interval observer

Model-based fault detection is based on generating a residual comparing the measurements of physical variables $\boldsymbol{y}(k)$ of the system with their estimation $\hat{\boldsymbol{y}}(k)$ provided by a model:

$$\boldsymbol{r}(k) = \boldsymbol{y}(k) - \hat{\boldsymbol{y}}(k) \tag{5}$$

According to (Gertler 1998) and considering the input/output observer expression given by Eq.(3), the computational form of a residual generator is given by

$$\boldsymbol{r}(k, \boldsymbol{\theta}) = -\boldsymbol{G}(q^{-1}, \boldsymbol{\theta})\boldsymbol{u}(k) + (I - \boldsymbol{H}(q^{-1}, \boldsymbol{\theta}))\boldsymbol{y}(k) \tag{6}$$

where: $\boldsymbol{r}(k) \in \Re^{ny}$ is the residual (or ARR) set.

Then, when considering model uncertainty located in parameters, the residual generated by (6) will not be zero even in a non-faulty scenario. In this case, the possible values of this residual could be bounded using an interval (Puig et al. 2002)

$$r_i^o(k) \in [\underline{r}_i^o(k), \overline{r}_i^o(k)] \tag{7}$$

where:

$$\underline{r}_i^o(k) = \underline{\hat{y}}_i(k) - \hat{y}_i^o(k) \ \text{ and } \ \overline{r}_i^o(k) = \overline{\hat{y}}_i(k) - \hat{y}_i^o(k) \tag{8}$$

are computed considering the nominal observer output prediction $\hat{\boldsymbol{y}}^{\boldsymbol{o}}(k)$ obtained using $\boldsymbol{\theta} = \boldsymbol{\theta}^o \in \boldsymbol{\Theta}$ and the $[\,\underline{\hat{y}}(k)\,,\,\overline{\hat{y}}(k)\,]$ given by (4).

This residual interval provides an ***adaptive threshold***. When condition (7) is not fulfilled, a fault is indicated by the interval observer.

Degree of residual violation

As it is proposed in (Puig et al. 2005), the activation value for each residual is calculated as in the DMP-approach (Petti et al., 1990) using the Kramer function:

$$\phi_i(k) = \begin{cases} \dfrac{(r_i^o(k)/\overline{r}_i^o(k))^4}{1+(r_i^o(k)/\overline{r}_i^o(k))^4} & \text{if} \quad r_i^o(k) \ge 0 \\ \\ -\dfrac{(r_i^o(k)/\underline{r}_i^o(k))^4}{1+(r_i^o(k)/\underline{r}_i^o(k))^4} & \text{if} \quad r_i^o(k) < 0 \end{cases} \tag{9}$$

In this way, residuals are normalized to a metric between -1 and 1, $\phi_i(k) \in [-1, 1]$, which indicates the degree to which each equation is satisfied: 0 for perfectly satisfied, 1 for severely violated high and -1 for severely violated low.

Fault residual sensitivity concept

The ***sensitivity of the residual*** (Gertler 1998) to a fault is given by

$$\boldsymbol{S}_f(q^{-1}) = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{f}} = \boldsymbol{G}_f(q^{-1}) \tag{10}$$

where $\boldsymbol{G}_f(q^{-1})$ is the transfer function that describes the effect on the residual, $\boldsymbol{r}$, of a given a fault, $\boldsymbol{f}$.

In the following, Eq. (10) is particularized to an output / input sensor fault as these are the most usual faults in an urban sewer system.

When these kinds of faults are considered, the measurements given by those sensors can be written as it follows:

$$\begin{aligned} \boldsymbol{y}(k) &= \boldsymbol{y}_0(k) + \boldsymbol{F}_y(\theta)\boldsymbol{f}_y(k) \\ \boldsymbol{u}(k) &= \boldsymbol{u}_0(k) + \boldsymbol{F}_u(\theta)\boldsymbol{f}_u(k) \end{aligned} \tag{11}$$

where $\boldsymbol{f}_u \in \Re^{nu}$ and $\boldsymbol{f}_y \in \Re^{ny}$ are the input/output sensor faults respectively being $\boldsymbol{F}_u(\boldsymbol{\theta})$ and $\boldsymbol{F}_y(\boldsymbol{\theta})$ their associated matrices while $\boldsymbol{u}_0$ and $\boldsymbol{y}_0$ are the real system input and output respectively. Thus, using Eq. (6) and Eq.(10), the residual sensitivity to an output fault is given by

$$\boldsymbol{S}_{fy}(q^{-1}, \boldsymbol{\theta}) = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{f}_y} = \left(\boldsymbol{I} - \boldsymbol{H}(q^{-1}, \boldsymbol{\theta})\right)\boldsymbol{F}_y(\boldsymbol{\theta}) \tag{12}$$

The sensitivity value at time instant *k=0*, i.e., when the fault appears is

$$\boldsymbol{s}_{fy}(0) = \boldsymbol{F}_y(\boldsymbol{\theta}) \tag{13}$$

independently of the observer gain matrix $\boldsymbol{W}$. That, means ***simulators***, ***observers*** and ***predictors*** have the same capabil-

ity to detect this kind of fault at its occurrence time instant. On the other hand, the steady-state value (Gertler 1998) for an abrupt fault modelled as a unit-step function is

$$\boldsymbol{s}_{fy}(\infty) = (\boldsymbol{I} - \boldsymbol{C}(\boldsymbol{\theta})(\boldsymbol{I} - \boldsymbol{A}_o(\boldsymbol{\theta}))^{-1}\boldsymbol{W})\boldsymbol{F}_y(\boldsymbol{\theta}) \quad (14)$$

In general, except the simulation approach, once the fault occurs, the residual capability to detect faults (fault residual sensitivity) decreases because of the use of the measurements to correct the model estimations.

In the case of the residual sensitivity to an input sensor fault, it is also obtained using Eq. (6) and Eq. (10)

$$\boldsymbol{S}_{fu}(q^{-1},\boldsymbol{\theta}) = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{f}_u} = -\boldsymbol{G}(q^{-1},\boldsymbol{\theta})\boldsymbol{F}_u(\boldsymbol{\theta}) \quad (15)$$

The residual fault sensitivity at time instant $k=0$ when the fault appears is

$$\boldsymbol{s}_{fu}(0) = \boldsymbol{0} \quad (16)$$

independently of the observer gain matrix $\boldsymbol{W}$. That means none input fault can be detected at this time instant. While its steady-state value for an abrupt fault modelled as a unit-step function is given by

$$\boldsymbol{s}_{fu}(\infty) = -\boldsymbol{C}(\boldsymbol{\theta})(\boldsymbol{I} - \boldsymbol{A}_o(\boldsymbol{\theta}))^{-1}\boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{F}_u(\boldsymbol{\theta}) \quad (17)$$

Table 1 describes the sensitivity dynamics in terms of $\boldsymbol{W}$.

	Simulation W=0	***Observation 0<WC< A***	***Prediction WC= A***
$\boldsymbol{S}_{fy}$	Constant	Pulse response	Deadbeat
$\boldsymbol{S}_{fu}$	Dynamic response	Dynamic response	Dynamic response

Table 1. Fault residual sensitivity dynamics in terms of observed gain matrix, $\boldsymbol{W}$.

Case of study: Barcelona urban drainage system

To illustrate the approach proposed in this paper, a real case study based on the Barcelona urban drainage system is used. In particular, the main focus is on detecting faults in the limnimeters used to measure the sewer levels. The sewer network is modelled using a simplified graph relating the main sewers and a set of virtual and real reservoirs (Cembrano et al. 2002). A ***virtual reservoir*** is an aggregation of a catchment of the sewer network which approximates the hydraulics of rain, runoff and sewage water retention using a mass balance:

$$\frac{dV(t)}{dt} = Q_{up}(t) - Q_{down}(t) + I(t)S \quad (18)$$

where: V is the volume of water accumulated in the catchment, Q_{up} and Q_{down} are flows entering and exiting the catchment, I is the rain intensity falling in the catchment and S its surface. Input and output sewer levels are measured using limnimeters and they can be related with flows using a linearised Manning relation: $Q_{up}(t) = M_{up}L_{up}(t)$ and $Q_{down}(t) = M_{down}L_{down}(t)$. Moreover, it is assumed that: $Q_{down}(t) = K_v V(t)$. Then, substituting those relations in Eq. (18) and considering that the measurement sampling time is T_s = 300 s, the following discrete-time model for each limnimeter can be obtained:

$$L_{down}(k+1) = aL_{down}(k) + bL_{up}(k) + cI(k)) \quad (19)$$

where: $a = (1 - K_v\Delta t)$, $b = M_{up}K_v\Delta t / M_{down}$ and $c = SK_v / M_{down}$. Using this modelling methodology, the model of the considered part of the Barcelona's sewer network is presented in Fig. 1.

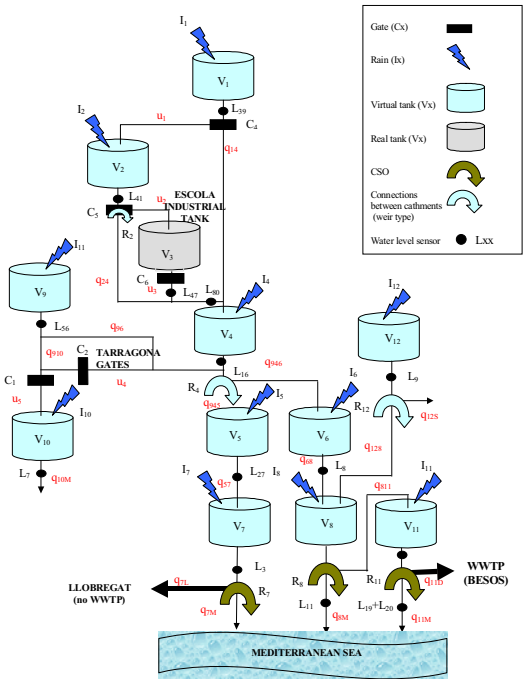


Fig. 1. Model of the considered part of the Barcelona Network

In this paper, only the residual set associated to the limnimeters L_{03} and L_{27} will be considered to illustrate the obtained results. This residual set is composed of two residuals, one for each limnimeter. Thus, using the limnimeters models given by Eq. (19), the observers used to monitor L_{03} and L_{27} are given by

$$\hat{L}_{03}(k) = a_{03}(1-\lambda_{03})\hat{L}_{03}(k-1) + b_{03}L_{27}(k-1) + c_{03}I_{14}(k-1) + \\ + a_{03}\lambda_{03}L_{03}(k-1) \quad (20)$$

$$\hat{L}_{27}(k) = a_{27}(1-\lambda_{27})\hat{L}_{27}(k-1) + c_{27}I_{20}(k-1) + a_{27}\lambda_{27}L_{27}(k-1) \quad (21)$$

where λ_{03} and λ_{27} are the associated observer gains using the parameterizations $w_{03}=\lambda_{03}a_{03}$ and $w_{27}=\lambda_{27}a_{27}$ (λ_{03} =0, simulation; λ_{03} =1, prediction). I_{14} and I_{20} are the rain intensities measured by the rain gauges P_{14} and P_{20}, respectively. The model parameters are obtained using parameter estimation from experimental data. Considering that the uncertainty associated to the following intervals describes the possible values of each parameter: a_{03}=*[0.8816, 0.9084]*, b_{03}=*[0.0381, 0.0393]*, c_{03}=*[1.4469e4, 1.4910e4]*, a_{27}=*[0.9544, 0.9737]* and c_{27}=*[6.2641e3, 6.3907e3]*.

The nominal residual associated to limnimeter L_{03} using the observer (20) is given by

$$r^o_{03}(k)=\frac{1-a^o_{03}q^{-1}}{1+a^o_{03}(\lambda^o_{03}-1)q^{-1}}L_{03}(k)-\frac{b^o_{03}q^{-1}}{1+a^o_{03}(\lambda^o_{03}-1)q^{-1}}L_{27}(k) \quad (22)$$
$$-\frac{c^o_{03}q^{-1}}{1+a^o_{03}(\lambda^o_{03}-1)q^{-1}}I_{14}(k)$$

Then, considering Eq. (12), the sensitivity of this residual to an additive fault in L_{03} ($\boldsymbol{F}_y(\boldsymbol{\theta})=\boldsymbol{I}$) is

$$S^o_{03}(q^{-1})=\frac{1-a^o_{03}q^{-1}}{1+a^o_{03}(\lambda^o_{03}-1)q^{-1}} \quad (23)$$

that satisfies that in a faulty scenario, the residual is activated from the occurrence time instant t_0 if the fault is detected, according to Eq. (13). On the other hand, in case of an additive fault affecting L_{27} ($\boldsymbol{F}_u(\boldsymbol{\theta})=\boldsymbol{I}$) and considering Eq. (15), the sensitivity of the residual associated to the limnimeter L_{03} is

$$S^o_{03-27}(q^{-1})=-\frac{c^o_{03}q^{-1}}{1+a^o_{03}(\lambda^o_{03}-1)q^{-1}} \quad (24)$$

According to Eq. (16), this residual is never activated at t_0 as its sensitivity is null at this time instant.

Analogously, in case of the limnimeter L_{27}, its nominal residual is

$$r^o_{27}(k)=\frac{1-a^o_{27}q^{-1}}{1+a^o_{27}(\lambda^o_{27}-1)q^{-1}}L_{27}(k)-\frac{c^o_{27}q^{-1}}{1+a^o_{27}(\lambda^o_{27}-1)q^{-1}}I_{20}(k) \quad (25)$$

Thus, the sensitivity of this residual to an additive fault affecting L_{27} is

$$S^o_{27}(q^{-1})=\frac{1-a^o_{27}q^{-1}}{1+a^o_{27}(\lambda^o_{27}-1)q^{-1}} \quad (26)$$

while its sensitivity to an additive fault affecting L_{03} is

$$S^o_{27-03}(q^{-1})=0 \quad (27)$$

As in the case of limnimeter L_{03}, the residual sensitivity given by Eq.(26) shows the residual is activated from the occurrence time instant t_0 if the fault is detected.

Integration between fault detection and fault isolation modules

The residual generator given by Eq. (6) is the key element which links the fault detection stage with the fault isolation stage. From Eq. (10), the fault residual sensitivity can be obtained which determines when a fault is detected and the time the fault indication is hold. On the other hand, the residual generator stores the influence that a given fault has on the residual set. This knowledge allows the fault diagnosis algorithm to isolate the fault, when the residual set has different properties associated to each fault. Finally, since the residual generator expression given by Eq. (6) depends on the observation gain matrix $\boldsymbol{W}$, the performance of the fault detection and isolation modules also depends on $\boldsymbol{W}$.

Fault isolation algorithm: FSM matrices

Objective

In (Meseguer et al. 2006c), four different fault signature matrices are considered: Boolean fault signal activation (***FSM***01), fault signal signs (***FSMsign***), fault residual sensitivity (***FSMsensit***), and fault signal occurrence order (***FSMorder***). Those matrices store the influence of the considered faults on the residual set: the element $\boldsymbol{FSM}_{ij}$ of a matrix contains the expected influence of fault f_j on $r_i{}^0$. In this paper, a new fault property is introduced: the fault signal occurrence time (***FSMtime***). This section focuses on how to determine ***FSMsensit*** and ***FSMtime*** and how they are affected by the observer gain matrix $\boldsymbol{W}$ using the fault sensitivity residual concept. The other matrices can be derived from them.

In the original algorithm, the first component between the fault detection and fault isolation modules is an interface which is based on a memory that stores along a time window given by T_w, and for each residual, the time instant ($k_{\phi i}$) in which the residual has been activated, ($|\phi_i(k)|\geq 0.5$) Eq. (9), and the activation value (ϕ_{imax}) whose absolute value is maximum. Then, at the end of the considered time window, an isolation result is given based on the memory stored information and on a pattern comparison component. While at least one of the residuals is activated, Eq. (9), ($|\phi_i(k)|\geq 0.5$), the pattern comparison component compares at every time instant the memory information with the information stored in the theoretical fault signature matrices what allows to reject those fault hypotheses which are not consistent with the observed activation values. At the end of the time window T_w or when just one fault hypothesis is left, a fault isolation result is given. T_w must be long enough so that all the residuals could have been activated during at least one time instant along this time window whatever the fault is. This paper shows T_w can be obtained from the matrix ***FSMtime***

Finally, it must be taken into account that the influence of a fault affecting the output sensor y_i on its associated residual r_i is registered in the diagonal element $\boldsymbol{FSM}_{ii}$ of the considered fault signature matrix.

FSMsensit: Evaluation of fault sensitivities

The fault residual sensitivity describes the residual capacity to detect faults. Eq. (6) shows that residual property has a time evolution and is influenced by the observer gain. The following equations describe how to compute the entries $\boldsymbol{FSM}sensit_{ij}$

$$\boldsymbol{FSM}sensit_{ij}=\begin{cases}\dfrac{S_{f_{ij}}(q^{-1})\eta(k-t_0)}{\left|r^o_i(k)\right|} & if \quad r^o_i(k)\geq 0 \text{ and } k\geq t_0\\ \dfrac{S_{f_{ij}}(q^{-1})\eta(k-t_0)}{\left|r^o_i(k)\right|} & if \quad r^o_i(k)<0 \text{ and } k\geq t_0\\ 0 \quad if \quad k<t_0 \text{ or } S_{f_{ij}}(q^{-1})=0\end{cases} \quad (28)$$

where $\eta(k)$ is an unitary abrupt step input, $S_{f_{ij}}$ is the sensitivity associated to the nominal residual $r^o_i(k)$ regarding the fault hypothesis f_j and t_0 is the fault occurrence time instant. When t_0 is unknown, it must be estimated using the

activation time instant $k_{\phi i}$ of the first activated residual. As consequence of the fault residual sensitivity time dependency, $\boldsymbol{FSM}sensit_{ij}$ evolves dynamically since the fault occurrence time instant t_0. This function is corrected using the detection threshold as the bigger it is, the more difficult the residual activation is, according to Eq. (7).

Focusing on sensor faults and on the residual generator described by Eq. (6), in general, the next information is derived regarding the fault isolation stage:

- Eq. (13) shows if an output sensor fault is detected, it must be from t_0. Therefore, when an output sensor fault occurs, the fault occurrence time instant t_0 is always known since it is equal to the activation time instant $k_{\phi i}$ of its associated residual.
- Eq. (16) shows the residual sensitivity to an input sensor fault is null at t_0 and thus, that fault can not be detected at this time instant.

The evaluation component computes $\boldsymbol{factor}sensit_j$ for the j^{th}-fault hypothesis using the $\boldsymbol{FSM}sensit$ table for weighting the activation values $\phi_i(k)$ (Meseguer et al. 2006c):

$$\boldsymbol{factor}sensit_j(k) = \frac{\left|\sum_{i=1}^{n}\left(\phi_i(k)\boldsymbol{FSM}sensit_{ij}\right)\right|}{\sum_{i=1}^{n}\left|\boldsymbol{FSM}sensit_{ij}\right|}\boldsymbol{zvf}_j \quad (29)$$

where:

$$\boldsymbol{zvf}_j = \begin{cases} 0, & \text{if } \exists\ i \in \{1,\ldots,n\} \text{ with } \boldsymbol{FSM}\text{sensit}_{ij} = 0 \\ & \text{and } |\phi_i(k)| \geq 0.5 \\ 1, & \text{otherwise} \end{cases} \quad (30)$$

where the factor $\boldsymbol{zvf}_j$ excludes those fault hypothesis that has a zero theoretical sensitivity while fault signal presents an non-zero value.

Case of study: *FSMsensit* Considering the residual generator associated to the limnimeters L_{03} and L_{27} (Eq. (22) and Eq. (25)), the corresponding matrix *FSMsensit* is analyzed in this *Section*.

The elements of matrix $\boldsymbol{FSM}sensit$ are calculated dynamically at every time instant once a fault is detected (Eq. (28)). The observer gains have an influence on the elements of $\boldsymbol{FSM}sensit$ which might help to differentiate the effect a fault produces in the residual set. Concerning the case of study and according to Eq. (28), the elements of this matrix are given by *Table 2* when k $\geq t_0$ and $r_{03}{}^0$ (Eq.22), $r_{27}{}^0$ (Eq.25) are positive.

	f(L03)	f(L27)
r(L03)	$\frac{S_{03}^0(q^{-1})\eta(k-t_0)}{\left\|\overline{r}_{03}^o(k)\right\|}$	$\frac{S_{03-27}^0(q^{-1})\eta(k-t_0)}{\left\|\overline{r}_{03}^o(k)\right\|}$
r(L27)	0	$\frac{S_{27}^0(q^{-1})\eta(k-t_0)}{\left\|\overline{r}_{27}^o(k)\right\|}$

Table 2. FSMsensit Matrix

FSMtime: Evaluation of fault signal occurrence time

When a fault f_j appears, the affected residuals need different times to start indicating that fault. Each element of the j^{th} column associated to the matrix $\boldsymbol{FSM}time$ contains the time activation interval $[\underline{\varphi}_{ij}, \overline{\varphi}_{ij}]$ in which the residual is expected to start indicating that fault. The value $\overline{\varphi}_{ij}$ is associated to the minimum f_j-type fault which is considered to be isolated, while $\underline{\varphi}_{ij}$ is associated to the maximum f_j-type fault the monitored system might suffer. Thus, according to (Meseguer et al. 2006b), the values $\underline{\varphi}_{ij}$ for a given fault f_j could be estimated carrying out a test for every residual. Derived from Eq. (7), this test can be written as:

$$S_{f_{ij}}(q^{-1})f_j^*(k)q^{-t_0} \notin [\underline{r}_i^o(k), \overline{r}_i^o(k)] \quad k \geq t_0 \quad (31)$$

where $f_j^*(k)$ is the worst case of a f_j-type fault the monitored system might suffer. Then, the time the residual requires to start indicating the fault ($\underline{\delta}_{ij}$) is obtained using the minimum time instant k_{min} that satisfies Eq. (31).

$$\underline{\delta}_{ij} = k_{\min} - t_0 \quad (32)$$

When monitoring a system, the fault occurrence time instant t_0 is unknown in general. Hence, the values $\underline{\delta}_{ij}$ associated to the fault hypothesis f_j must be referred to the first activated residual. Then,

$$\underline{\varphi}_{ij} = \underline{\delta}_{ij} - \min_{\forall i}(\underline{\delta}_{ij}) \quad (33)$$

The value $\overline{\varphi}_{ij}$ might also be calculated using test (31) but in this case, $f_j^*(k)$ is the minimum f_j-type fault which is considered to be isolated. Thus, the values $\overline{\delta}_{ij}$ are obtained and then,

$$\overline{\varphi}_{ij} = \overline{\delta}_{ij} - \min_{\forall i}(\underline{\delta}_{ij}) \quad (34)$$

Regarding the elements of matrix $\boldsymbol{FSM}time$,

$$\boldsymbol{FSM}time_{ij} = \begin{cases} [\underline{\varphi}_{ij}, \overline{\varphi}_{ij}] & if \quad S_{f_{ij}}(q^{-1}) \neq 0 \\ [-1,-1] & if \quad S_{f_{ij}}(q^{-1}) = 0 \end{cases} \quad (35)$$

it is remarkable the influence of the observer gain on the interval $[\underline{\varphi}_{ij}, \overline{\varphi}_{ij}]$ and consequently, a proper design might help so that all the residuals were activated at the same time instant. When just sensor faults are considered the residual sensitivity functions given by Eq. (12) and Eq. (15) allow establishing some conditions concerning to the residual activation time instant as it was mentioned before. Then, the elements of matrix $\boldsymbol{FSM}time$ can be expressed as it follows:

$$\boldsymbol{FSM}time_{ij} = \begin{cases} [0,0] & if \quad i = j \\ [\underline{\varphi}_{ij}, \overline{\varphi}_{ij}] & if \quad i \neq j \text{ and } S_{F_{ij}}(q^{-1}) \neq 0 \\ [-1,-1] & if \quad S_{F_{ij}}(q^{-1}) = 0 \end{cases} \quad (36)$$

assuming the model associated to each system output is not static. Thus, in order to compare that information with the

stored in ***FSMtime,*** the ***factor***$time_j$ is calculated as it follows:

$$\boldsymbol{factor}time_j(k) = \frac{\sum_{i=1}^{n}\left(ckecktime\left(k_{\phi i}, k_{ref}, \boldsymbol{FSM}time_{ij}\right)\right)}{\sum_{i=1}^{n} boolean(\boldsymbol{FSM}time_{ij})} \boldsymbol{zvf}_j \quad (37)$$

where $k_{\phi i}$ is the activation time instant of the i^{th} residual, k_{ref} is the activation time instant of the first activated residual,

$$ckecktime\left(k_{\phi i}, k_{ref}, \boldsymbol{FSM}time_{ij}\right) = \begin{cases} 0 & if \quad k_{\phi i} - k_{ref} \notin \boldsymbol{FSM}time_{ij} \text{ or } |\phi_i(k)| < 0.5 \\ 1 & if \quad k_{\phi i} - k_{ref} \in \boldsymbol{FSM}time_{ij} \text{ and } |\phi_i(k)| \geq 0.5 \end{cases} \quad (38)$$

$$boolean(\boldsymbol{FSM}time_{ij}) = \begin{cases} 0, & \text{if } \boldsymbol{FSM}time_{ij} = [-1,-1] \\ 1, & \text{if } \boldsymbol{FSM}time_{ij} \neq [-1,-1] \end{cases} \quad (39)$$

Case of study: *FSMtime*. In this *Section*, considering the residual generator associated to the limnimeters L_{03} and L_{27} (Eq. (22) and Eq. (25), matrix *FSMtime* is analyzed.

In this case, the expression (36) can be used to calculate the ***FSM***time matrix elements. Thus, all elements of this matrix are determined except for the element $\boldsymbol{FSM}time_{12}$ which is calculated dynamically when the first residual is activated. When test (31) is carried out, a simplification is done regarding the considered $[\underline{r}_i^o(k), \overline{r}_i^o(k)]$: a constant adaptive threshold is used whose value is given by the worst case regarding the fault detection. This is

$$\left[-\tau_i, \tau_i\right] \quad (40)$$

where $\tau_i = \max\left\{\max_{\forall k \geq t_0}(abs(\underline{r}_i^o(k))), \max_{\forall k \geq t_0}(abs(\overline{r}_i^o(k)))\right\}$

where *abs* is the absolute value function.

The worst sensor fault of type f_j is assumed to be an abrupt step whose gain $f_j^{\max}$ is equal to the maximum estimated value of all sensors affected by this type of fault. The minimum fault of type f_j which is considered to be isolated is assumed to be an abrupt step whose gain $f_j^{\min}$ is derived from the minimum detectable fault functions of type f_j associated to each residual. This gain is calculated as it follows:

$$f_j^{\min} = \max_{\forall i}\left\{abs(f_{ij}^{\min})\right\} \quad (41)$$

where $f_{ij}^{\min}$ is the steady-state value of the minimum detectable fault (Meseguer et al. 2006a) of type f_j regarding the i^{th} residual considering its associated threshold is given by the Eq. (40).

$$f_{ij}^{\min} = \frac{\tau_i}{s_{fij}(\infty)} \quad (42)$$

where $s_{fij}(\infty)$ is the steady-state value of the fault residual sensitivity given by Eq.(14) or Eq.(17) depending on the sensor type (input or output).

In the following, the values in seconds of the ***FSM***time matrix associated to residuals (22) and (25) are shown for two specific scenarios: *Scenario 1* (Table 3.a) where ***t_0=4000 s,*** $\lambda_{03}^o = 0.01$, $\lambda_{27}^o = 0.01$ and ***Scenario 2 (Table 3.b)*** where ***t_0=4000*** , $\lambda_{03}^o = 0.01$, $\lambda_{27}^o = 0.5$

	f(L03)	f(L27)
r(L03)	[0,0]	[900,6300]
r(L27)	[-1,-1]	[0,0]

	f(L03)	f(L27)
r(L03)	[0,0]	[900,3900]
r(L27)	[-1,-1]	[0,0]

Table 3. ***FSM***time Matrix a) for $\lambda_{03}^o = 0.01$, $\lambda_{27}^o = 0.01$ and b) $\lambda_{03}^o = 0.01$, $\lambda_{27}^o = 0.5$

These tables confirm the influence of the observer gain on matrix ***FSM***time.

Diagnosis result: Logic decision

The last task of the considered diagnosis algorithm is the decision logic component. At the end of the time window T_w, this element gives a diagnosis result based on the calculated factors associated to each fault hypothesis f_j: ***factor***$sensit_j$, ***factor***$time_j$. This result indicates the most probable fault which is affecting to the considered system.

Influence of the observer gain on the diagnosis result

Such as it was mentioned previously, the residual generation in fault detection stage and the fault influence on the residuals stored in the theoretical fault signature matrices depend on the observer gain matrix ***W***. Thus, all the isolation factors (***factor***$sensit_j$, ***factor***$time_j$) also depend on the observer gain ***W*** since they are obtained from the activation values ϕ_i given by Eq. (9) and from the matrices ***FSM***$sensit$ and ***FSM***$time$. Therefore, because the diagnosis result is a function of these factors, this stage is also influenced by the observer gain ***W***.

Decision logic algorithm

In this *Section*, the decision logic component is introduced in order to give an idea of how it works in spite it is not the aim of this paper. The decision logic algorithm starts when the first residual is activated ($|\phi_i(k)| \geq 0.5$) at time instant $k_{\phi i}$ and lasts T_w time instants or till all fault hypothesises except one are rejected because they do not fulfil the observed residual activation order or because an unexpected activation signal has been observed according to those fault hypotheses. Regarding the time window T_w, it must be calculated so that once the first residual is activated, the rest of the residuals can be activated, at least, during one time instant whatever the fault is and thus, according to ***FSM***$time$

$$T_w = \max_{\forall i,j}(\overline{\varphi}_{ij}) \quad (43)$$

At the end of the time window T_w, for each non-rejected fault hypothesis, a probability is calculated using the factors ***factor***$sensit_j$ and ***factor***$time_j$. Thus, the biggest probability will determine the diagnosed fault. The probability associated to the fault hypothesis f_j is determined as it follows:

$$d_j = \max(\boldsymbol{factor}sensit_j, \boldsymbol{factor}time_j) \quad (44)$$

In this algorithm, ***factor***$time_j$ plays a significant role on the final result as this factor is very robust regarding the

non-persistence fault indication given by the detection stage while ***factor***$sensit_j$ is not so robust. On the other hand, if the fault diagnosis result might be very influenced by the model errors, both factors should be used since ***factor***$time_j$ might be inaccurate.

Case of study: Fault isolation scenarios

In this section, considering the residual generator associated to the limnimeters L_{03} and L_{27} (Eq. (22) and Eq. (25)), the time evolution of the residuals and the fault isolation factors (***factor***$sensit_j$ and ***factor***$time_j$) are plotted considering the two scenarios introduced in the last *Section* when a constant additive fault appears at time instant t_0=*4000 s* affecting L_{27}. The value of this fault is equal to the minimum isolable fault according to the *Scenario 1* $f_{L_{27}} = 0.5566$ m. The difference between these scenarios is only the values of the observer gains in order to show their influence and the importance of ***factor***$time_j$ when the result given by ***factor***$sensit_j$ is confusing. In those *Scenarios*, the factors are calculated at every time instant k (T_w=*300 s* corresponding to one sample since T_s=*300 s*) since then, the effect of the observer gains on them can be seen more clearly. Besides, a third *Scenario* is considered which is like *Scenario 2* but where the right value of T_w is used according to *Table 3.b* and Eq. (43).

Scenario 1: t_0=4000 s, T_w=300 s, $\lambda^o_{03} = 0.01$ ***and*** $\lambda^o_{27} = 0.01$

In this fault scenario, the fault is clearly detected and isolated such as it is seen according to the time evolution of the residuals (Fig. 2) and the fault isolation factors (Fig. 3). It can be noticed from Fig. 2, the fault is detected at its occurrence time instant by the residual associated to L_{27} according to its sensitivity (Eq. (26)) while the residual associated to L_{03} does not detect it till later such as it was expected according to its sensitivity and the values of matrix ***FSM***$time$.

Finally, it should be taken into account that while only one residual is activated, only ***factor***$sensit_j$ reaches its maximum value as the residual sensitivity of L_{03} is nearly zero-valued when fault appears. This is because the residual sensitivities are dynamical functions.

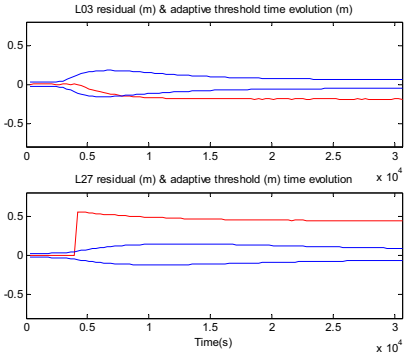


Fig. 2.Time evolution of the residuals and their adaptive thresholds in scenario 1.

Scenario 2: t_0=4000 s, T_w=300 s, $\lambda^o_{03} = 0.01$ ***and*** $\lambda^o_{27} = 0.5$

In this scenario once the fault occurs, it is only clearly indicated by the residual associated to L_{27} during few time instants (Fig. 4). Concerning the other residual, the fault is detected such as it was in the *Scenario 1*. This inaccurate behaviour of the fault detection module causes ***factor***$sensit_j$ to be also inaccurate regarding the fault isolation result (Fig. 5). Conversely, because the residual activation time fulfils the fault pattern stored in matrix ***FSM***$time$, the factor ***factor***$time_j$ is not affected by this lack of fault indication and it let clearly isolate the fault.

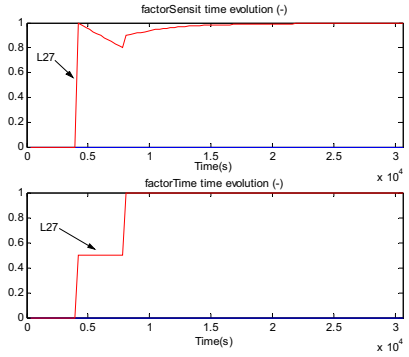


Fig. 3. Time evolution of the factor indicators in scenario 1.

Scenario 3: t_0=4000 s, T_w=3900 s, $\lambda^o_{03} = 0.01$ ***and*** $\lambda^o_{27} = 0.5$

Unlike *Scenario 2*, in this *Scenario* the right value of the time window T_w, according to Eq. (43) is used: T_w=*3900 s*. This value assures that once the first residual is activated ($r_{27}{}^0$ at t_0=*4000 s*: see *Fig. 4*), the rest of the residuals ($r_{03}{}^0$) could be activated, at least, during few time instants at the end of this time window (t=t_0+T_w=*7900s*). Then, the first diagnosis result is given which already let isolate clearly the fault (Fig. 6). This result is kept constant till the end of the next time window.

Conclusions

Fault isolation is influenced by fault detection as both of them use the residual generator as a key element. It allows the fault isolation algorithm to know the effect that the considered faults have on the residual set, determining the theoretical fault signature matrices. Thus, because the residual generator depends on the observer gain ***W***, the influence of this matrix on the fault isolation module is also concluded.

This work follows the trend in FDI to analyze the influence of the detection stage in the isolation results (Combastel et. al 2003). The main contribution is the method provided for on-line computation of the ***FSM*** including temporal information about symptom evolution –i.e. residual activations-, which can be used to support or reject fault modes in the isolation stage. Moreover, related to recent works in BRIDGE, the interval model-based observer which is used in behaviour estimation for detection purposes provides a different way to cope with uncertainty in model parameters: modal intervals (Armengol et al. 2001), or different techniques for envelope generation (Loiez et al. 1998;Rinner et al., 2004). Finally, from a pure DX perspective, this work

can be seen as an initial step towards on-line calculation of temporal symptom evolution as needed in diagnosis chronicles (Cordier et al. 2000). As a further work, the concept of minimum isolable fault would be introduced and a new version of the fault isolation algorithm is planned to be proposed using the results of this paper.

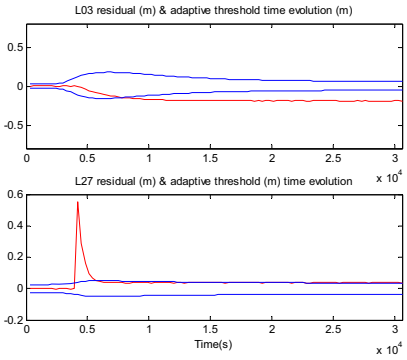


Fig. 4.Time evolution of the residuals and their adaptive thresholds, scenarios 2 and 3.

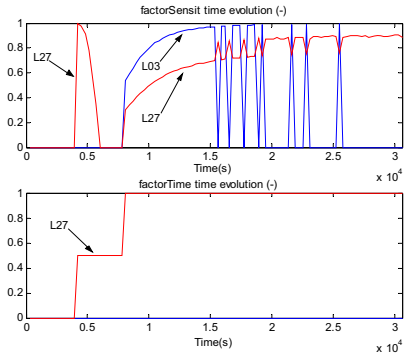


Fig. 5. Time evolution of the factor indicators in scenario 2.

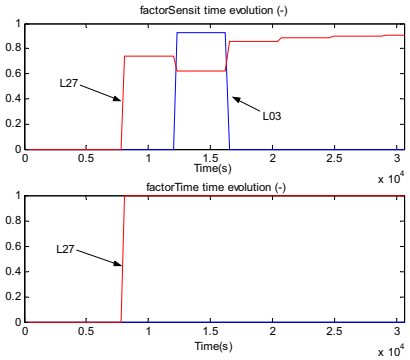


Fig. 6. Time evolution of the factor indicators in scenario 3.

References

Armengol, J., Vehí, J., Travé-Massuyès, L., Sainz, M.A. "Application of multiple sliding time windows to fault detection based on interval models" Workshop on Principles of Diagnosis DX'01, pp. 9-16. San Sicario, Italy, 2001.

Cembrano, G. "Global Control of the Barcelona Sewerage System for Environment Protection". In Proceedings of IFAC World Congress, Barcelona, 2002.

Chow, E., Willsky, A. 1984. "Analytical redundancy and the design of robust failure detection systems". IEEE Transactions on Automatic Control, Volume: 29 , Issue: 7 , July , Pages: 603 – 614.

Combastel, C., S. Gentil, and J. P. Rognon. 2003. "Toward a better integration of residual generation and diagnostic deci-sion". IFAC SAFEPROCESS'03, Washington, USA.

Cordier, M.O. C. Dousson. 2000 « Alarm driven monitoring based on chronicles«. IFAC SAFEPROCESS'00. p. 286-291. Budapest, Hungary.

Gertler, J. 1998. Fault Detection and Diagnosis in Engineering Systems. M. Dekker.

E. Loiez, P. Taillibert, 1997. "Polynomial temporal band sequences for analog diagnosis". In Proc. of IJCAI-97. Pgs. 474-479. Nagoya, Japan.

Meseguer, J., Puig, V., Escobet, T. 2006a "Observer gain effect in linear observer-based fault detection". IFAC SAFEPROCESS'06. Beijing. China.

Meseguer, J., Puig, V., Escobet, T. 2006b. "Observer gain effect in linear interval observer-based fault detection" IFAC SAFEPROCESS'06. Beijing. China.

Meseguer, J., Puig, V., Escobet, T., Quevedo, J. 2006c. "Observer gain effect in linear interval observer-based fault isolation" Workshop on Principles of Diagnosis DX'06. Peñaranda de Duero. Spain.

Petti, T.F., J. Klein, and P. S. Dhurjati. 1990. "Diagnostic model processor: Using deep knowledge for process fault diagnosis," AIChE Journal, vol. 36, p. 565.

Ploix, S., Adrot, O. and J. Ragot. "Parameter Uncer-tainty Computation in Static Linear Models". 38th IEEE Conference on Decision and Control. Phoenix. Arizona. USA.

Puig, V., Quevedo, J., Escobet, T., De las Heras, S. 2002. "Robust Fault Detection Approaches using Interval Models". IFAC World Congress (b'02). Barcelona. Spain.

Puig, V., Saludes, J., Quevedo, J. 2003. "Worst-Case Simulation of Discrete Linear Time-Invariant Dynamic Systems", Reliable Computing 9(4): 251-290, August.

Puig, V. J. Quevedo, T. Escobet, and B. Pulido. 2005. "A New Fault Diagnosis Algorithm that Improves the Integration of Fault Detection and Isolation" in ECC-CDC'05, Sevilla, Spain.

Puig, V., Quevedo, J., Escobet, T., Meseguer, J. 2006. "Toward a better integration of passive robust interval-based FDI algorithms". IFAC SAFEPROCESS'06. Beijing. China.

B. Pulido, V. Puig, T. Escobet, and J. Quevedo. 2005 "A new fault localization algorithm that improves the integration between fault detection and localization in dynamic systems". Workshop on Principles of Diagnosis DX'05. Monterey, California, USA.

B. Rinner, U. Weiss. 2004 "On-line monitoring by dynamically refining imprecise models" IEEE Trans. On SMC, Part B. Vol. 34, No. 4. Pg. 1811-1822. Special Section on Diagnosis of Complex Systems: Bridging the methodologies of the FDI and DX communities. August.

Diagnosis of Multi-Agent Plans under Partial Observability

Roberto Micalizio **Pietro Torasso**
Università di Torino, Dipartimento di Informatica
corso Svizzera 187, Torino, Italy; tel. (+39) 011 6706773 - fax. (+39) 011 751603
{micalizio,torasso}@di.unito.it

Abstract

The paper discusses a distributed approach for monitoring and diagnosing the execution of a plan where concurrent actions are performed by a team of cooperating agents.

The notions of plan diagnosis and agent diagnosis are introduced to model primary and secondary failures and the root causes of the primary failures respectively. The paper addresses the problem of diagnosing multi-agent plans when the partial observability of the status of the system does not allow to univocally determine the outcome of all the actions executed so far. In particular, we presents a methodology where the detection of the outcome of *target actions* activates a diagnostic process, aimed at estimating the outcome of other actions previously executed and at detecting, if any, primary failures. Such diagnostic inferences are then propagated to the other agents for determining the secondary failures.

In the monitoring and diagnostic steps, the system makes use of a relational model of the actions for capturing both the nominal and the abnormal result of their execution.

Introduction

In the AI community it is widely recognized that multi-agent systems constitute a powerful approach for solving complex tasks in a distributed way. Distributing a task among cooperating agents of a team offers several advantages: the agents can pursue different goals in parallel, distributed resources may be used more efficiently, and multi-agent systems are more robust since the risk of a global failure is reduced.

However, distributed problem solving presents a number of challenges to deal with (see e.g. (Carver & Lesser 2003) in particular, the execution of a multi agent plan (MAP) can be *threatened* (Birnbaum *et al.* 1990) by the occurrence of unexpected events, such as faults in the agents or harmful interactions which arise when a number of agents try to use the same resources simultaneously. Although in some particular cases it is reasonable to assume that the planner is able to prevent all kinds of threats (Boutilier & Brafman 2001), this is in general unfeasible (think for example to the need of anticipating all the possible occurrences of faults that may affect the functionalities of the agents).

The occurrence of plan threats does not necessarily imply that the plan goal can no longer be achieved. In many cases, in fact, the plan goal can still be achieved but the plan may need to be adjusted (i.e. a recovery process is required). In order to repair a multi-agent plan, one has to know not only which actions are failed (*primary failures*), but also how the rest of the plan has been affected by these failures (*secondary failures*). In other words, the execution of a multi-agent plan needs to be *on-line monitored* and *diagnosed* to take into account possible anomalous evolutions during the execution of the plan.

It is worth noting that a MAP is substantially different from a component-based system. The behavior of a component-based system essentially depends on the health status of its components and on the commands submitted to them. The relations among the system components (e.g., the topological architecture of the system) usually do not change over time. Therefore the model of a component-based system can be defined a-priori and does not change during the diagnosis.

On the contrary, the behavior of a MAP depends both on the actions the agents execute and on the occurrence of faults in the functionalities of these agents. As a consequence, the relations among the agents (e.g., competition or cooperation) change over time and the model of the MAP needs to be adjusted according to the current set of actions.

Moreover, also the notion of diagnosis of a MAP is different from the notion of diagnosis of a component-based system. In fact, the classical notion of model-based diagnosis of a component-based system consists of a (minimal) set of system components which are assumed faulty to explain the anomalous observed behavior of the system itself. On the contrary, as discussed in (Witteveen *et al.* 2005), the notion of diagnosis of a MAP consists of a (minimal) subset of actions executed by the agents, whose failure explains the observed behavior.

Finally, one of the peculiar characteristics of a MAP is represented by the fact that the agents of the team typically adopt a cooperative behavior. In general, the cooperation consists of an agent i that provides another agent j with some services. In these cases, a fault in the functionalities of agent i, causing the failure of the current action of i (*primary failures*), may cause the failure of some actions assigned to j (*primary failures*), which in turn may have cascade effects on the actions assigned to other agents in the team.

After this premises it is evident that the model-based methodologies developed to monitoring and diagnosis a

component-based system are just partially applicable when the system under consideration is a MAP. In this paper we discuss and formalize a novel model-based methodology to solve the problems of monitoring and diagnosing the execution of a MAP with atomic actions. In particular we propose a distributed methodology, where each agent is responsible for monitoring and diagnosing the sub-plan it executes. We assume that actions are atomic and we model them as relations, which capture their effects both in the nominal and in the anomalous situations.

As one of the main contributes of this work, we provide two notions of diagnosis: the notion of *plan diagnosis* highlights which primary and secondary failures have occurred, while the notion of *agent diagnosis* explains every primary failure in terms of a combination of faults occurred in the functionalities of an agent; moreover the paper points out the relation between the agent and the plan diagnosis.

As a further contribution, the methodology formalized in this paper is able to deal with *weakly observable systems*; under the conditions of weak observability an agent may be unable to determine the outcome of every action it executes. This means that an agent must be able to exploit the observations available at a given time to determine, if possible, the outcome of some actions previously executed.

The paper is organized as follows. In the following section we introduce the basic notions of global and local plans, then we formalize the processes of monitoring and diagnosis of a MAP. The paper closes with some concluding remarks.

Setting the Framework

The class of systems we deal with is referred to as *synchronous transition systems* (Kurien & Nayak 2000). In these systems the time is a discrete sequence of instants, the actions are executed synchronously by the agents in the team and each action in P takes a time unit to be executed (this is a common assumption, see e. g. (Witteveen *et al.* 2005)); finally, at each instant some observations are available (typically the system is just partially observable).

The problem of supervising (monitoring + diagnosing) the execution of P is decomposed into a set of sub-problems: the global plan P is decomposed in as many sub-plans as the agents in the team $\mathcal{T}$; each sub-plan P_i is assigned to agent i, which is responsible for executing and supervising the actions in P_i. At each time instant t, an agent i receives a set of observations obs_t^i relevant for the status of agent i itself, although in general the observations obs_t^i are not sufficient for precisely inferring the status of agent i.

The partitioning of activities among agents described above does not guarantee that the sub-problems are completely independent of one another. In fact, we deal with the case where the agents in $\mathcal{T}$ cooperate to achieve a common global goal G. The cooperation among agents consists in the exchange of services, as will be formalized in the following of this section. In particular, given an agent i and its next action a, a is *executable* iff the set of preconditions $pre(a)$ are satisfied in the current status estimated by i; some of the preconditions in $pre(a)$ may be services, which other agents in $\mathcal{T}$ have to provide; that is, there exists a causal link $l : a' \xrightarrow{q} a$, where $a' \in A_j$. When agent i intends to execute action a, the service q may be not provided yet, in this case we assume that agent i voluntarily waits for the satisfaction of q. In other words, agent i slightly adjusts its sub-plan P_i by adding a *Wait* action. On the other hand, when agent j is able to determine whether the service q has been achieved or not (i.e., agent j determines the outcome of action a'), it notifies the waiting agent i about the status of the service q; in this way we avoid that agent i waits indefinitely for the required services. To keep the discussion simple, in this paper the outcome of an action can be either *succeeded*, when the action is executed in the nominal way, or *failed* otherwise.

It is worth noting that the cooperative behavior among agents introduces causal dependencies among the actions the agents have to execute; therefore, when an action failure occurs in the sub-plan P_i (i.e., a *primary failure* occurs), the failure may affect the sub-plans of other agents, namely, the failure may propagate not only in P_i, but even in the global plan P. According to the traditional terminology (see e.g., (Pencolé & Cordier 2005)), the actions which fail as a direct or indirect consequence of a primary failure are called *secondary failures*.

Albeit the supervision of the execution of plan P is decomposed into a set of sub-problems, these sub-problems can not be solved independently of one another. In fact, the agents need to communicate during the supervision process, in order to determine, in a distributed way, a plan diagnosis which highlights the distinction between primary and secondary failures. Moreover, we formalize the relation between primary failures and agent diagnoses and show the critical role played by the agent diagnosis to determine the set of primary failures.

The structure of a Multi-Agent Plan. In this section we briefly introduce some basic notions on the multi-agent plan. In particular, we assume that the global plan P to be monitored is synthesized by means of a planner similar to the partial-order planner (POMP) by Boutilier and Brafman (Boutilier & Brafman 2001). The POMP planner considers atomic, STRIPS-like actions (i.e. an action a is modeled in terms of its preconditions *pre*(a) and effects *effects*(a)); the STRIPS language is augmented with a *concurrent* clause which specifies, for each action a, a list of actions of other agents which can, or can not, be executed concurrently with a. During the planning process, the POMP keeps track of two kinds of links between actions:

- *causal* links, e.g. $a \xrightarrow{q} a'$ indicates that the action a provides the action a' with the service q, where q is an atom occurring in the preconditions of a';

- *precedence* links, e.g. $a \prec a'$ indicates that the action a must precede the execution of the action a'. These links are used to resolve agents competitions: when two (or more) agents need to access the same resource, the POMP produces a plan where the access to the resource is serialized.

We require that the global plan P satisfies the following characteristics:

- the actions assigned to an agent are totally ordered

- the precedence links among actions of different agents are translated into causal links where the offered service consists in relinquishing a resource. For example, the precedence link $a \prec a'$ rules the access to the resource *res* be-

tween the action a, assigned to agent i, and a', assigned to agent j. This precedence link is replaced by the causal link $a \xrightarrow{free(res)} a'$ to make clear that role played by the service *free(res)*which enables action a' since *free(res)* appears in the preconditions of a'.

Formally, the global plan P is a tuple $\langle A, <, C\rangle$, where A is the set of actions the agents have to execute, $<$ is an order relation between actions assigned to the same agent, C is a set of causal links of the form $a \xrightarrow{q} a'$ where a and a' may or may not be assigned to the same agent.

Main services and target actions Given an agent $i \in \mathcal{T}$ and an action a assigned to i (i.e., $a \in A_i$), *main*(a) denotes the set of main services produced by a, more precisely each effect $q \in$*main*(a) satisfies at least one of the following conditions:

- $q \in G$ i.e., q is an atom which appears in the goal G of the global plan.

- $q \neq$ *free(res)* and it is a service agent i provides to another agent j, that is, there exists a causal link $a \xrightarrow{q} a'$ where $a' \in A_j$.

An action $a \in A_i$ such that *main*$(a) \neq \emptyset$ is said a *target action*, the action is said *simple* otherwise. Given an action a, either target or simple, *depends_on*(a) is the set of simple actions $\{a_1, \ldots, a_n\}$ in A_i, which directly or indirectly provide a with a service; i.e., for each action a_h in $\{a_1, \ldots, a_n\}$ there exists a sequence of causal links in C_i from a_h to a. Possibly, *depends_on*$(a) = \emptyset$ only when a is a target action.

Distributed Control Loop. As stated in the introduction, we propose a distribute approach where each agent i of the team $\mathcal{T}$ is responsible for monitoring and diagnosing the actions it performs summarized in the architecture reported in Figure 1. The plan instance P, achieving a desired complex goal is synthesized by a human user possibly by exploiting planning tools (such as (Boutilier & Brafman 2001)). Given the global P, the Dispatcher module decomposes P in as many sub-plans as the agents in $\mathcal{T}$. The decomposition is an easy task, which involves just the selection from P of all the actions an agent has to execute; formally, the sub-plan for agent i is the tuple P_i=$\langle A_i, <_i, C_i, X_i^{in}, X_i^{out}\rangle$ where: A_i is the subset of actions in A that agent i has to execute; $<_i$ is a total order relation defined over the actions in A_i; C_i is a set of causal links $a \xrightarrow{q} a'$ where both a and a' belong to A_i; X_i^{in} is a set of *incoming* causal links where $a' \in A_i$ and $a \in A_j$; finally, X_i^{out} is a set of *outgoing* causal links.

After this initial phase, each agent starts the execution of its own sub-plan. In particular, each agent performs a local control loop on the progress of its actions. This local control loop involves the Plan Execution Monitoring and Diagnosis module (PEMD), the Fault Recovery module (FR) and the Local Re-plan module (LP). First of all, the PEMD module is responsible for estimating the current status of the agent and for detecting the *outcome* of the actions the agent is executing. To this aim the PEMD exploits the observations obs_t^i coming from the environment.

Every time agent i detects a not nominal outcome, it tries to recover from this failure by activating a local re-planning phase aimed at synthesizing a local recovery plan. In particular, in the current architecture of the supervisor, the FR module coordinates the recovery process: first, the FR invokes the LP module, which has to infer a new local plan that agent i can execute to overcome the failure. In case such a local plan does not exist, agent i notifies the human user about the detected action failure.

MOVE(A, P1, P2)	LOAD(A, OBJ, P)	PUT_DOWN(A, OBJ, P)
pre: AT(A, P1)	**pre**: AT(A, P)	**pre**: AT(A, P)
FREE(P2)	AT(OBJ, P)	LOADED(A, OBJ)
	EMPTY(A)	
eff: ∼AT(A, P1)	**eff**: ∼EMPTY(A)	**eff**: ∼LOADED(A, OBJ)
∼FREE(P2)	∼AT(OBJ, P)	AT(OBJ, P)
AT(A, P2)	LOADED(A, OBJ)	EMPTY(A)

Table 1: The preconditions and the effects of the actions in the plan of Figure 2.

While in (Micalizio, Torasso, & Torta 2006a) we have addressed the FD and the LP modules, which are aimed at the plan recovery from an action failure; in this paper we focus our attention on the PEMD module, devoted to the monitoring and the diagnosis of the execution of the local plan P_i. Observe that, since actions may be causally dependent, the PEMD modules of different agents may need to communicate during the plan supervision process (see the communication channel between the PEMD modules in Figure 1).

Running Example. In the paper we will use a simple example from the blocks world for illustrating the concepts and the techniques introduced for monitoring and diagnosing the execution of a MAP. Let us consider two agents, `A1` and `A2`, which have to cooperate in order to achieve a specific configuration where the block `B1` is put on the block `B2`. To get this goal, agent `A1` has to move block `B1` from the source `S1` to the target `T1` by passing through door `D1`, while agent `A2` has to move block `B2` from `S2` to `T1` by passing through the doors `D2` and `D1`.

The use of the resources (source and target locations and doors) is constrained since a resource can be accessed by only one agent per time. However, there exists a further location, called `PRK` (i.e. parking area), which is not constrained and where the agents are positioned when they complete their sub-plans.

Table 1 reports the STRIPS-like definition of a sub-set of actions the agents can execute. For example, the action `LOAD(A,OBJ,P)` means that the agent `A` loads on board the object `OBJ` located at position `P`. The preconditions of this action require that both `A` and `OBJ` are located at `P` and that the agent is not loaded yet with another object. The effects of action `LOAD(A,OBJ,P)` state that the object is loaded on `A`.

Figure 2 shows the global plan produced by a POMP-like planner to achieve the target configuration of blocks, i.e. the global goal. The plan is a DAG where nodes correspond to actions and edges correspond to temporal (dashed) or causal (solid) links. The causal links are labeled with the services an action provides to another one, for example the causal link from action 1 to action 4 is labeled with the service `LOADED(A1,B1)`, which is both one of the effects of action 1 and one of the preconditions for the execution of action 4. It is easy to see that when action 1 fails, action 4 fails too since one of its preconditions is not satisfied.

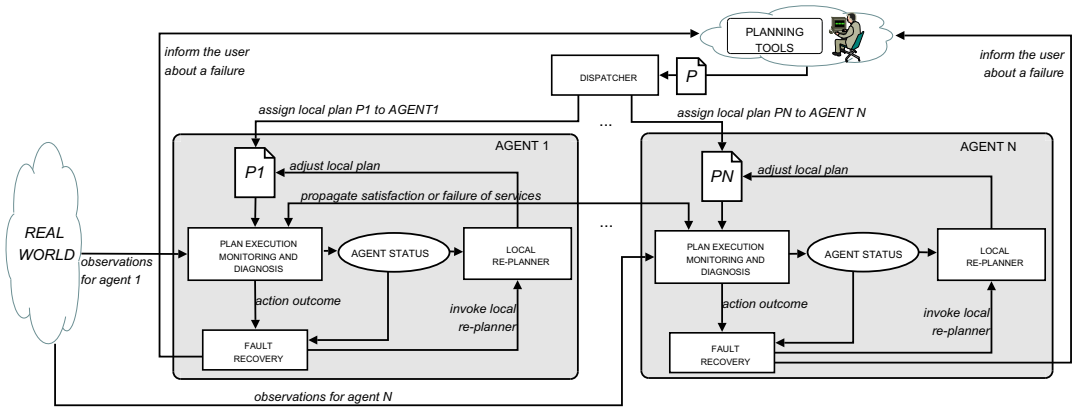


Figure 1: The architecture of the distributed control loop.

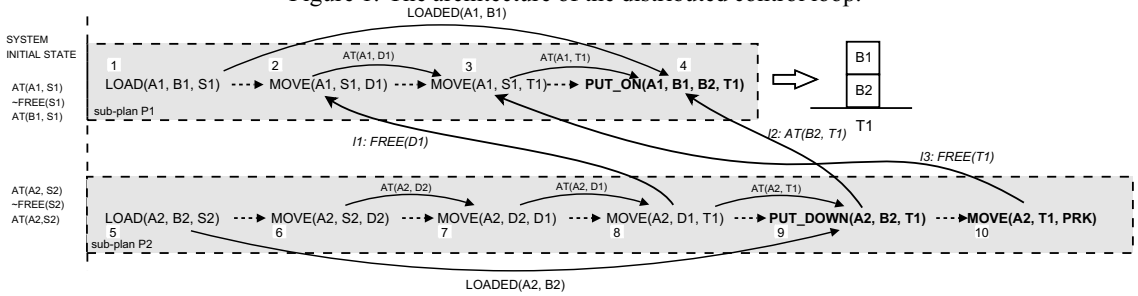


Figure 2: The plan built by a POMP-like planner.

In Figure 2 the target actions are in bold, for example given the target action 4, the set *depends_on*(4) is $\{1, 2, 3\}$. The grey, dashed rectangles highlight the sub-plans in which the global plan has been decomposed. The presence of the causal links l_1, l_2 and l_3 indicates that the sub-plans P_1 and P_2 have causal dependent actions.

Plan Execution Monitoring: Basic Concepts

Since each agent i has to supervise the actions it is responsible for, agent i has to determine, at each time t, the outcome of the action a_t (i.e., the action executed by i at time t), and its own health status after the execution of a_t.

Agent Status. The status of agent i is expressed in terms of a set of status variables VAR^i; in particular, this set of variables is partitioned into two subsets: $HEALTH^i$ and $ENDO^i$. $HEALTH^i$ denotes the set of variables concerning the health status of an agent functionalities (e.g., mobility and power), these variables are not directly observable, so their actual value can be just estimated. Whereas, $ENDO^i$ denotes the set of all the other endogenous status variables (e.g., the agent's *position*). The values assumed by the variables in $ENDO^i$ can be directly observed by agent i; however, since agent i receives just partial observations about the changes occurring in the system, at each time instant t the agent i observes the value of just a subset of variables in $ENDO^i$. Therefore, agent i is in general unable to precisely determine its own status; rather i can determine just a set of states after the execution of a_t, this set is known in literature as *belief state* and will be denoted as $\mathcal{B}_t^i$.

Extended Action Model. As mentioned above, the synthesis of the plan P can be performed by exploiting STRIPS-like action models, which consider only the nominal behavior of the actions. However, in order to supervise the execution of the actions in P_i, the agent i must have at disposal extended action templates, which model not only the nominal behavior of the actions, but also the actions behavior when faults occur. For such a modeling task we adopt a relational representation, which has been proved to be useful for the on-line monitoring and diagnosis of multi-agent systems (see (Micalizio, Torasso, & Torta 2006b)). In particular, this formalism is able to capture non deterministic effects of the action execution.

The extended model of an action a_t is a transition relation $\Delta(a_t)$, where every tuple $d \in \Delta(a_t)$ models a possible change in the status of agent i, which may occur while i is executing a_t. More precisely, each tuple d has the form $d = \langle s_{t-1}, fault, s_t \rangle$; where s_{t-1} and s_t represent two agent states at time $t-1$ and t respectively, each state is a complete assignment of values to the status variables VAR^i of agent i; $fault$ denotes the fault which occurs to cause a change of status from s_{t-1} to s_t (of course, in the transitions which model the nominal behavior $fault$ is empty). As commonly assumed in approaches to the diagnosis of Discrete Events Systems (see e.g. (Pencolé & Cordier 2005)), we assume that two faults can not occur simultaneously on the functionalities of the same agent. It follows that an agent

i can be struck by only one fault per time instant, however it can be affected by more than one faults over the time; moreover, the assumption allows the simultaneous occurrence of faults in different agents.

Table 2 shows the extended model of a MOVE action from a location $P1$ to a location $P2$; due to space reasons we show just a subsets of tuples of the transition relation $\Delta(\texttt{MOVE})$. Moreover, for sake of readability, the agent states are expressed just in the subset of status variables directly affected by the MOVE action, in particular they are: *position*, the position of the agent; *carried* the carried load (if any) by the agent, this variable can be either *empty* or *loaded*; *pwr*, the power of the agent's battery, which can be *high* (the nominal mode) or *low* (a degraded mode) and *mobility*, the health status of the agent's mobility functionality which can be *ok* (nominal) or *bk* (i.e., broken, a faulty mode). Observe that the STRIPS model of the MOVE action just represents the nominal behavior of MOVE, assuming that both power and mobility are in their nominal mode; the corresponding extended model represents also how the MOVE action behaves in any combination of power and mobility modes, included the faulty ones.

The nominal behavior of the MOVE action is modeled by transition 1, the $*$ symbol (don't care) indicates that, when the agent's functionalities behave nominally, the actual carried load does not impact the outcome of the action. In our example the MOVE action can fail as a consequence of two types of faults: f-BRY and f-MOB. f-BRY affects the battery by reducing the level of power from the nominal *high* to the degraded *low*. An agent can complete a move action with battery power *low* iff the agent is not carrying any block, the action fails otherwise (see transitions 2 and 3). f-MOB affects the health status of the mobility functionality, which changes from the nominal *ok* to the anomalous *bk* (broken); under this health status there is no way to complete the move action (see transition 4).

Action Outcome. Since the actual execution of an action is threatened by the occurrence of unexpected events (e.g., faults), the outcome of an action may be not nominal. The outcome of an action is *succeeded* when all the effects of the action have been achieved, or *failed* otherwise. Since at each time instant t we may have to deal with an ambiguous belief states $\mathcal{B}_t^i$, we consider the action a as succeeded only when all the expected effects of action a hold in every state s included in $\mathcal{B}_t^i$; more formally.

Definition 1 *The outcome of action a_t, executed by i at time t, is* succeeded *iff $\forall q \in eff(a_t)$, $\forall s \in \mathcal{B}_t^i$, $s \models q$; i.e., iff all the atoms q in $eff(a_t)$ are satisfied in every state s in $\mathcal{B}_t^i$.*

Of course, when we can not assert that action a_t is succeeded we assume that the action is failed.

Agent Trajectory. One of the results of the monitoring performed by agent i over the execution of its actions is a trajectory of the status of i itself over time. In general, the trajectory of a system is the sequence of states, actions and observations generated from a given initial state to some time point of interest. However, since the system is only partially observable, agent i can not maintain the actual trajectory of its own state; rather agent i maintains a set of possible trajectories. In the following $Tr_i[0,t]$ will denote the set of trajectories tr maintained by i in the interval $[0,t]$.

Monitoring with uncertain action outcomes

In this section we describe the methodology for monitoring the execution of a MAP when the outcome of some actions may be uncertain. The algorithm implementing the approach is reported in Figure 3. Due to space reasons, we focus just on the basic concepts the algorithm relies on.

The prediction process. In order to monitor the execution of action a_t, the first step performed by agent i is to extend the set of trajectories $Tr_i[0,t-1]$ with all possible state transitions of action a_t. In particular the prediction process is formalized in terms of the relational operators as:

Definition 2 *Given the set of trajectories $Tr_i[0,t-1]$ and the extended model of the action a_t, Tr_i is incrementally extended as: $Tr_i[0,t]$=$\sigma_{obs_t^i}(Tr_i[0,t-1] \bowtie \Delta(a_t))$*

The join operator $Tr_i[0,t\text{-}1] \bowtie \Delta(a_t)$ represents the prediction step, it has the effect of adding, to each trajectory $tr \in Tr_i[0,t-1]$, a possible state of agent i after the execution of action a_t. This set of predictions is filtered out w.r.t. the observations available at time t by means of the selection $\sigma_{obs_t^i}$ (see line 8 of the algorithm of Figure 3).

Observe that $Tr_i[0,t]$ can be extensionally written as:
$\sigma_{obs_t^i}(\sigma_{obs_{t-1}^i}(\ldots(\sigma_{obs_1^i}(\mathcal{B}_0^i \bowtie \Delta(a_1))) \ \ldots \ \bowtie \ \Delta(a_{t-1})) \bowtie \Delta(a_t))$
where $\mathcal{B}_0^i$ represents the initial belief state of agent i.

Given $Tr_i[0,t]$, agent i is able to extract by means of the relational projection the current belief state $\mathcal{B}_t^i$; in this way agent i evaluates the outcome of action a_t according to Definition 1; formally, $\mathcal{B}_t^i$=$\pi_t(Tr_i[0,t])$.

Monitoring and system observability. If the system were completely observable, the predicted belief state $\mathcal{B}_t^i$ would contain just the actual status of agent i at time t. Of course, this condition is rather unrealistic and, in general, one has to deal with uncertain belief states. On the contrary, some minimal requirements on the system observability have to be met to provide agent i with the possibility of evaluating the outcome of the actions it has to execute.

Basically we consider two possible levels of system observability: *strong* and *weak*. Strong observability corresponds to the assumption that agent i is able to determine the outcome of every action a_t it executes; i.e., the belief state $\mathcal{B}_t^i$, filtered out w.r.t. obs_t^i, is sufficiently precise for testing the condition of Definition 1.

Dealing with uncertain outcomes. Unfortunately, the conditions of strong observability do not hold in many domains. In the general case, the estimated belief state $\mathcal{B}_t^i$ is not sufficiently precise to determine the outcome of action a_t. In other words, it may be possible that the effects of a_t are satisfied in just a not empty subset of states in $\mathcal{B}_t^i$.

In this case agent i has to keep track of the actions it has executed and whose outcome has not been determined yet. In the following, $pendingOutcomes_i(t)$ denotes the set of actions, so far executed by agent i, whose outcome has not been determined at time t. In other words, at each time instant t, either agent i is able to determine the outcome of a_t or a_t is included in the set $pendingOutcomes_i(t)$.

	s_{t-1}				$fault$	s_t			
	endogenous		**health variables**			**endogenous**		**health variables**	
	position	*carried*	*power*	*mobility*		*position*	*carried*	*power*	*mobility*
1	P1	*	High	OK	*null*	P2	*	High	OK
2	P1	EMPTY	High	OK	f-BRY	P2	EMPTY	Low	OK
3	P1	LOADED	High	OK	f-BRY	P1	LOADED	Low	OK
4	P1	*	High	OK	f-MOB	P1	*	High	BK

Table 2: The extended model of action MOVE(A,P1,P2): the transition relation $\Delta(MOVE)$.

Plan-Monitoring-and-Diagnosis($\mathcal{B}_0^i$, P_i) [P_i=$\langle A_i, <_i, C_i, X_i^{in}, X_i^{out}\rangle$] {
01 t = 0; t_s = 0; $Tr_i[t_s,t] = \mathcal{B}_0^i$; *pendingOutcomes*$_i(t) = \emptyset$; *nextAction* = null;
02 **while** (true){ $t = t + 1$;
03 **if** (*nextAction*=null) *nextAction* = ⟨*get next action from* P_i⟩;
04 **if** (*all incoming links of nextAction are marked as satisfied*) mark *nextAction* as *executable*;
05 **else** ⟨ execute *WaitAction* ⟩;
06 **if** (*nextAction* is *executable*) {
07 a_t=*nextAction*; *nextAction*=null; ⟨execute a_t⟩; obs_t^i = ⟨*get current observations for agent i*⟩;
08 $Tr_i[t_s,t]$=$\sigma_{obs_t^i}(Tr_i[t_s,t\text{-}1]\bowtie\Delta(a_t))$=$\sigma_{obs_t^i}(\sigma_{obs_{t-1}^i}(\ldots(\sigma_{obs_{t_s+1}^i}(\mathcal{B}_{t_s}^i\bowtie\Delta(a_{t_s+1})))\ldots\bowtie\Delta(a_{t-1}))\bowtie\Delta(a_t))$;
09 $pendingOutcomes_i(t) = pendingOutcomes_i(t-1)\cup\{a_t\}$;
10 $\langle completedActs_i, primaryFailedActs_i\rangle$=**EvaluateOutcomes**($pendingOutcomes_i(t)$, $Tr_i[t_s,t]$);
11 **for** (each action $a \in completedActs_i$){
12 **for** (each link $l : a \xrightarrow{q} a' | l \in C_i$) mark l as *satisfied*;
13 **for** (each link $l : a \xrightarrow{q} a' | l \in X_i^{out}$) notify *agent*($a'$) the satisfaction of link l; }
14 $AgentDiagnosis_i = \emptyset$; $secondaryFailedActs_i$=$\emptyset$; *Inbox*=⟨get incoming messages⟩;
15 **for** (each satisfied link $l : a' \xrightarrow{q} a''$, $l \in$ *Inbox* | $l \in X_i^{in}$) mark l as *satisfied*;
16 **if** ($primaryFailedActs_i \neq \emptyset$) {
17 *ACTS* = $\emptyset$; *LNKS* = $\emptyset$; $AgentDiagnosis_i$ = **getAgentDiagnosis**($primaryFailedActs_i$, $Tr_i[t_s,t]$);
18 **for** each action $a \in primaryFailedActs_i$ {
19 **Propagate**(a, P_i, *ACTS*, *LNKS*); $secondaryFailedActs_i = secondaryFailedActs_i \cup ACTS$
20 **for** (each failed link $l' : a'' \xrightarrow{q} a'''$, $l' \in LNKS$) notify *agent*(a''') the failure of l'; } } }
21 **repeat**
22 changes = false; *Inbox*=⟨get incoming messages⟩;
23 **for** (each failed link $l : a' \xrightarrow{q} a''$, $l \in$ *Inbox* | $l \in X_i^{in}$){ *LNKS*=$\emptyset$; *ACTS*=$\emptyset$; changes = true;
24 **Propagate**(a'', P_i, *ACTS*, *LNKS*); $secondaryFailedActs_i = secondaryFailedActs_i \cup ACTS$
25 **for** (each failed link $l' : a'' \xrightarrow{q} a'''$, $l' \in LNKS$) notify *agent*(a''') the failure of l'; }
26 **until** ($\sim$ changes)
27 **if** ($primaryFailedActs_i \neq \emptyset$) $\vee$ ($secondaryFailedActs_i \neq \emptyset$)
28 **return** ⟨ $AgentDiagnosis_i$, $primaryFailedActs_i$, $secondaryFailedActs_i$⟩;
29 **else if** (a_t *is a target action*) { $pendingOutcomes_i(t) = \emptyset$; $\mathcal{B}_t^i = \pi_t(Tr_i[t_s,t])$; $t_s = t$; $Tr_i[t_s,t] = \mathcal{B}_t^i$ } } }

Figure 3: The algorithm for monitoring and diagnosing the execution of a local plan.

It is worth noting that the outcome of a_t can be exploited to determine the outcome of some actions in $pendingOutcomes_i(t)$, in fact the following property holds.

Property 1 *Let a_t be an action (either target or simple) with outcome* succeeded*, then all the actions a_h in $pendingOutcomes_i(t) \cap depends\_on(a_t)$ have outcome* succeeded *too.*

However, when an action a_t fails, agent i may be unable to determine whether failure is due to a fault during the execution of a_t or a_t is failed as a consequence of the failure of other actions i in *depends_on*(a_t). In this case, in order to be conservative, we introduce the following policy.

Policy 1 *Let a_t be an action (either target or simple) with outcome* failed*, then all the actions a_h in $pendingOutcomes_i(t) \cap depends\_on(a_t)$ are marked* failed *too.*

At line 10 of the algorithm, function **EvaluateOutcomes** evaluates the outcomes of the actions in $pendingOutcomes_i(t)$ and returns the pair $\langle completedActs_i, primaryFailedActs_i\rangle$, where in $completedActs_i$ ($primaryFailedActs_i$) the *succeeded* (*failed*) actions are collected.
In any case, whenever agent i determines the outcome of an action a, it has to propagate the effects of this outcome in the global plan according to the following policies:

Policy 2 Communicate Positive Information *if a is* succeeded*, $\forall l : a \xrightarrow{q} a' | l \in X_i^{out}$, notify $agent(a')$ that the service q has been provided.*

Policy 3 Communicate Negative Information *if a is* failed*, $\forall l : a \xrightarrow{q} a' | l \in X_i^{out}$, notify $agent(a')$ that the service q can not be provided.*

Policies 2 and 3 are implemented, respectively, in lines 11-13 and 18-20 of the algorithm.
Running Example. Let us assume that in the blocks world example, under weak observability, the LOAD actions are

not observable; moreover the MOVE actions are observable only when an agent gets the target T1 or the parking area PRK. Thereby, the outcome of actions 1, 2, 5, 6, 7 can not be determined just relying on the system observations. Let us assume that at time t, agent 2 executes action 8. Observe that at time t, $pendingOutcomes_2(t)$=$\{5, 6, 7\}$; while $pendingOutcomes_1(t)$=$\{1\}$. If the outcome of action 8 is *succeeded*, agent 2 determines that actions 6 and 7 are *succeeded* too; moreover it informs agent 1 that the service *free(D1)* has been provided. Notice that, to determine the outcome of action 5, agent 2 needs to know the outcome of the target action 9. On the contrary, if the outcome of action 8 is failed, agent 2 informs agent 1 that service *free(D1)* can not be provided and activates the diagnostic process.

Agent Diagnosis and Plan Diagnosis

The presence of causal links between actions introduces dependencies even among different sub-plans. As a consequence, the failure of an action (*primary failure*) may affect many other actions in the global plan (*secondary failures*).

The distinction between primary and secondary failures is an important piece of information, which must be considered in the notion of *plan diagnosis*. In fact, in order to recover from the failure of a set of actions in the plan, one needs to know: 1) which actions are the primary failures and 2) which combination of faults in the functionalities of the agents may have caused these failures, i.e. the *root causes* of these failures.

Agent Diagnosis. As described in the previous section, as soon as agent i detects the failure of action a_t, i starts a diagnostic reasoning step in order to determine the root causes of this failure (see line 17 of the algorithm). It is worth noting that, the failure of action a_t may be a consequence of 1) the occurrence of a fault during the execution of a_t; or 2) the failure of an action a_h, previously executed ($h < t$). In the latter case, the action a_h is included in the set $pendingOutcomes_i(t)$ and its (not nominal) outcome can not be unequivocally determined as the system is only partially observable.

When agent i is unable to distinguish between the two possible cases, i adopts a conservative behavior (according to Policy 1) and determines the primary failures as follows:

Definition 3 *Let a_t be an action executed by agent i at time t, let* failed *be the outcome of a_t. The set of primary failures related with the failure of a_t is the set of actions: $primaryFailedActs_i$= $(pendingOutcomes_i(t) \cap depends\_on(a_t)) \cup a_t$.*

Observe that action a_t is always included in the set of the primary failures.

Once the primary failures have been singled out, agent i has to infer a local diagnosis, where the root causes of each primary failure are highlighted.

Definition 4 *Let $a_h \in primaryFailedActs_i$ be the action executed at time h. The root causes of the failure of a_h is the set of faults $faults(a_h)$=$\pi_{faults}(\mathcal{B}_h^i \bowtie \Delta(a_h))$*

In this case the join $\mathcal{B}_h^i \bowtie \Delta(a_h)$ is an abductive step, since the relation $\Delta(a_h)$ is not used to predict a new belief state (as in the prediction process in Definition 2), but it is used for inferring the previous belief state $\mathcal{B}_{h-1}^i$. The projection π_{fault} collects the possible faults which may have caused a transition from a state in $\mathcal{B}_{h-1}^i$ to a state in $\mathcal{B}_h^i$, i.e., the result of the collection is the set of faults which may have occurred during the execution of action a_h. Since the failure of a_t can be explained by the failure of just one action in $primaryFailedActs_i$, the agent diagnosis for i is defined as:

Definition 5 *Given the set $primaryFailedActs_i$, the diagnosis for agent i $AgentDiagnosis_i$ is defined as $AgentDiagnosis_i = \bigvee_{a_h \in primaryFailedActs_i} faults(a_h)$*

In other words, $AgentDiagnosis_i$ is a disjunction of faults $f_1 \vee f_2 \vee \ldots \vee f_n$ affecting some functionalities of i, where each fault f_h is sufficient to explain the failure of an action in $primaryFailedActs_i$ and, as a consequence, of the action a_t. The inference of the agent diagnosis is part of the **getAgentDiagnosis** function (line 17).

Plan Diagnosis. Now we need to determine at what extent the primary failures affect the global plan P. In particular, we aim at singling out which actions in P won't become *executable* as a direct or indirect consequence of the failure of an action in $primaryFailedActs_i$. To this end we exploit the set C of causal links.

Definition 6 *Given the global plan P=$\langle A, <, C\rangle$ and the set $primaryFailedActs_i$ determined by agent i, the set of secondary failures is*
$secondaryFailedActs_i$=$\{a \in A$ such that
$\exists$ a causal link l' : $a' \xrightarrow{q'} a$, where $l' \in C$ and $a' \in primaryFailedActs_i$ or
$\exists$ a causal link l" : a" $\xrightarrow{q"} a$, where l" $\in C$ and a" $\in secondaryFailedActs_i\}$

Observe that, in order to single out the secondary failures in whole plan, the agents in the team have to cooperate. In particular, the propagation of a failure in the global plan is an iterative process in which the agents communicate one another the set of services which can longer not be provided. In the algorithm the process for computing the secondary failures is encoded in lines 21-26 of the algorithm and it is based on the **Propagate** function described in (Micalizio, Torasso, & Torta 2006a). In definitions 5 and 6 we have introduced the notion of plan diagnosis w.r.t. an agent i; however, these definitions can be extended to the more general case where the plan diagnosis considers all the agents in the team $\mathcal{T}$. More precisely, given the global plan P, it is possible to demonstrate that the plan diagnosis for P can be computed in distributed way as the pair $\langle primaryFailedActs, secondaryFailedActs\rangle$ where $primaryFailedActs = \bigcup_{i \in \mathcal{T}} primaryFailedActs_i$ and $secondaryFailedActs = \bigcup_{i \in \mathcal{T}} secondaryFailedActs_i$.

Correctness. The correctness analysis of the algorithm has to consider the prediction process and the inferences for computing the agent and plan diagnosis. Regarding the prediction process it is possible to demonstrate that the belief state $\mathcal{B}_t^i$, estimated at each time t, always contains the actual status of the agent i, i.e., the monitoring keeps track of the actual status of the agents. As concerns the agent diagnosis $AgentDiagnosis_i$, it always includes the actual set of faults which have caused a detected action failure

(this follows directly from the properties of the belief state $\mathcal{B}_t^i$). Finally, considering the plan diagnosis, it is easy to see that: 1) the set $primaryFailedActs_i$ actually contains all and only the actions whose failure may have caused the detected anomaly in the execution of the sub-plan P_i; and 2) the set $secondaryFailedActs_i$ includes all and only the actions in the global plan P which are affected by the failure of some actions in $primaryFailedActs_i$.

Moreover, assuming that the system observability is sufficient to determine unequivocally the outcome of every target action in the global plan P; and assuming that in each sub-plan P_i, the distance (in number of actions) between two consecutive target action is less than or equal to a constant k; it is possible to demonstrate that each agent $i \in \mathcal{T}$ is able to determine the outcome of all the actions it executes within a finite time window. This property allows agent i to maintain trajectories including just the last k time instants (in the algorithm, agent i maintains the trajectories in the interval $[t_s, t]$, where t_s is the time when the last target action has been successfully executed).

Running Example. Under the same conditions previously described, let us assume that at time t the failure of action 8 activates the diagnostic step. First of all, agent 2 determines the set of primary failures $primaryFailedActs_2 = \{6, 7, 8\}$. Then, agent 2 infers its local diagnosis, in this simple example $AgentDiagnosis_2$=$\{$`f-BRY` $\vee$ `f-MOB`$\}$. Finally, agent 2 propagates the harmful effects of the primary failures in the global plan in order to detect the secondary failures. In particular, the set $secondaryFailedActs_2$=$\{2, 3, 4, 9\}$ is obtained by means of the cooperation of agent 1, which propagates the failure of the causal link $l1$ in its own sub-plan.

Conclusions

While the problems of distributed monitoring and diagnosis of component-based systems have been deeply discussed in literature (see e.g., (Pencolé & Cordier 2005)); just a few solutions have been proposed in the multi-agent setting. In (Kalech & Kaminka 2005) the concept of *social diagnosis* is introduced; in particular they consider multi-agent systems where each agent chooses, at each time, the more appropriate behavior to assume. The social diagnosis has to explain the disagreements among cooperating agents (i.e., when some agents assume incompatible behaviors).

The approach presented in the paper is similar to the one described in (Witteveen *et al.* 2005), where Witteveen et al. propose a distributed approach to monitoring and diagnosing the execution of a MAP. This approach adopts an *object* (or *resource*) view since the status of the system is represented in terms of the status of the system objects. The approach assumes that every agent monitors and diagnoses every action it is responsible for. Actions are atomic and are modeled as functions of their nominal behavior only, whereas the anomalous behavior of the actions is not explicitly modeled. As a consequence, whenever an action failure is detected, the status of a subset of resources can not be predicted and the monitoring is incomplete.

In this paper, we have proposed a distributed approach for monitoring and diagnosing the execution of a multi-agent plan in a system which is only partially observable. In particular, while in (Micalizio, Torasso, & Torta 2006a) we have discussed a monitoring approach which assumes to determine the outcome of every action exploiting the available observations (strong observability) in this paper we have discussed an extension which is able to deal with uncertain action outcomes (weak observability).

Differently from the approach by (Witteveen *et al.* 2005), we introduce an extended model of actions for capturing both nominal and anomalous execution. Thereby the monitoring process we propose is complete since the status of the system is estimated even after the occurrence of a fault.

Moreover, while in (Witteveen *et al.* 2005) the notion of diagnosis refers just to the pair of primary and secondary failures, the use of extended action models allows us to introduce the notion of diagnosis which includes the root causes of the detected primary failures. Observe that the root causes are essential in order to recover from a failure, in fact, as discussed in (Micalizio, Torasso, & Torta 2006a), a local re-planner needs to know the current health status of an agent in order to establish which actions can or can not be used in the recovery plan. The extended model of the actions are encoded via OBDDs (Ordered Binary Decision Diagrams) and all the relational operations involved in the algorithm are performed by means of standard operations on OBDDs, according to the approach described in (Micalizio, Torasso, & Torta 2006b).

References

Birnbaum, L.; Collins, G.; Freed, M.; and Krulwich, B. 1990. Model-based diagnosis of planning failures. In *Proc. AAAI'90*, 318–323.

Boutilier, C., and Brafman, R. I. 2001. Partial-order planning with concurrent interacting actions. *JAIR* 14:105–136.

Carver, N., and Lesser, V. 2003. Domain monotonicity and the performance of local solutions strategies for cdps-based distributed sensor interpretation and distributed diagnosis. *Jour. of Aut. Agents and MAS* 6:35–76.

Kalech, M., and Kaminka, G. 2005. Diagnosing a team of agents: Scaling up. In *Proc. AAMAS'05*, 249–255.

Kurien, J., and Nayak, P. 2000. Back to the future for consistency-based trajectory tracking. In *Proc. American Association for Artificial Intelligence (AAAI00)*, 370–377.

Micalizio, R.; Torasso, P.; and Torta, G. 2006a. Intelligent supervision of plan execution in multi-agent systems. *Int. Trans. on Systems Science and Applications* 1(3):259–267.

Micalizio, R.; Torasso, P.; and Torta, G. 2006b. On-line monitoring and diagnosis of a team of service robots: a model-based approach. *AI Comm.* 19(4):313–349.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence* 164:121–170.

Witteveen, C.; Roos, N.; van der Krogt, R.; and de Weerdt, M. 2005. Diagnosis of single and multi-agent plans. In *Proc. AAMAS05*, 805–812.

Choosing Abstractions for Hierarchical Diagnosis

Fabien Perrot and **Louise Travé-Massuyès**
{perrot,louise}@laas.fr
0033561336952 0033561336302
LAAS-CNRS
Université de Toulouse
7, avenue du Colonel Roche,
31077 Toulouse Cedex 4
France

Abstract

This paper deals with the choice of abstractions for stating hierarchical diagnosis problems. Generally, hierarchical models are built manually and choosing the appropriate abstractions is quite an empirical science. To tackle this issue, we frame a diagnosis problem as an optimal constraint satisfaction problem (OCSP) and we define abstraction related to two OCSP's, in the structure and in the search space. This allow us to analyse the influence of the abstraction on the temporal computational complexity reduction offered by hierarchical reasoning. Optimal abstractions are shown to be built on the well-known diagnosis concept of potential conflict.

Introduction

In the artificial intelligence community, diagnosis problems are often formulated along the theory of consistency-based diagnosis (Hamscher, Console, & de Kleer 1992), one of the most widely used approaches to model-based diagnosis. However, such a logical formulation is being replaced more and more by a constraint optimisation formulation (Williams & Ragno 2003) which allows one to compare solutions. More precisely, a diagnosis problem is framed as an optimal constraint satisfaction problem (OCSP).

Abstraction has been advocated as one of the main remedies for the computational complexity of model-based diagnosis. It can be viewed as the process of generalisation by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose. Consequently, from a detailed diagnosis problem, one can construct by iterated abstractions more abstract diagnosis problems. These diagnosis problems can be solved together in hierarchical fashion (generally from the most abstract one to the least). Discovered solutions and non-solutions of the i^{th} diagnosis problem can be used to prune the search space of the $i-1^{th}$ diagnosis problem. Theoretically, temporal computational complexity reduction is exponential. In practice, this reduction notably depends on the choice of the abstract diagnosis problems, and thus depends on the choice of the abstractions. This reduction of complexity is observed in several research fields like symbolic model-checking, planning, constraint solving,.... It can be explained by the fact that abstraction is a way of reasoning about a set of objects (or states) satisfying specific properties rather than directly reasoning on the enumeration of objects. Such sets of states can be proved to be non-solution or solution, thus (if abstraction is well-chosen) it is a proof of solution or non-solution for all the belonging to the sets states. For example, the concept of conflict is by definition a direct proof that a set of states only contains non-solutions. It is more efficient to identify conflicts rather than testing and proving all the states one by one.

The aim of our work is to find, among all possible abstract diagnosis models, the one which gives the greatest complexity reduction. It would allow one to design and implement algorithms that build guaranteed "good" hierarchical diagnosis problems of a system, without the need for expertise. We are especially interested in abstraction techniques for CSP's in order to apply them to diagnosis. Abstraction of CSP's was recently surveyed in (Lecoutre *et al.* 2006) who proposed a new framework to describe general kinds of abstractions. Hierarchical diagnosis based on structural abstraction (which is a special case of abstraction) has been studied in detail in (Chittaro & Ranon 2004). Structural abstraction, from the diagnosis point of view, consists of clustering the components of the system. However, as far as we know, only one publication (Torta & Torasso 2006) in the diagnosis field, has been devoted to the choice of abstractions. The research in (Torta & Torasso 2006) deals with behavioural abstraction [1]. Our approach deals with more general kinds of abstractions.

The rest of this paper is organised as follows. In the second section, we succinctly present the OCSP framework and we recall how consistency-based diagnosis problems can be viewed in that framework. In the third section, we present the hierarchical diagnosis approach and the abstraction concepts. In the fourth and main section, we analyse the influence of the choice of abstractions on temporal computational reduction of hierarchical reasoning. Finally, we conclude and discuss future work.

OCSP's and diagnosis problems

The CSP framework

A constraint satisfaction problem (CSP) is defined as a set of variables and a set of constraints on these variables. Notably,

[1] The authors aggregate component modes but not combinations of modes, a special case of structural abstraction.

it is recognised by the following definition :

Definition 1 *A* CSP *is a pair* (V, C) *where :*

- $V = \{V_1, V_2, \ldots, V_n\}$ *is a set of* variables. *Each variable* V_i *has a non empty* domain $D(V_i)$ *of possible values.*
- $C = \{C_1, C_2, \ldots, C_m\}$ *is a set of* constraints. *Each constraint* C_j *involves some subset* $vars(C_j)$ *of* V *and specifies the allowable combinations of values for that subset.*

A state *of the problem is an assignment to all the variables in* V *:* $(V_1 = v^1, V_2 = v^2, \ldots, V_n = v^n)$. *A* partial state *of the problem is an assignment to only some variables (but not all) in* V. *A* consistent *assignment is one that does not violate any constraints. A* solution *to a CSP is a consistent state. A state is sometimes called a complete assignment and a solution is sometimes called a consistent and complete assignement.*

The structure of a CSP is given by a constraint hypergraph *where the nodes of the hypergraph correspond to variables of the problem and the hyperarcs correspond to constraints.*

The OCSP framework

An OCSP is a CSP for which one is allowed to compare solutions thanks to a cost function; and it is adapted to diagnosis in the sense that one is only interested in values of a subset of variables called decision variables. More formally :

Definition 2 *An* optimal constraint satisfaction problem (OCSP) *consists of a* $CSP = (V, C)$*, a set of* decision variables $W \subset V$*, and a* cost function $g : W \rightarrow \mathbb{R}^2$. *The remaining variables* $V - W$ *are called* non-decision variables. *An assignment of decision variables is called a decision state. A* solution *to an OCSP is a minimum cost decision state that is consistent with the CSP.*

For more details about OCSP's, please refer to (Williams & Ragno 2003).

The diagnosis problem

In the consistency-based diagnosis approach (Hamscher, Console, & de Kleer 1992), a model of the system to be diagnosed is used. This model is component-centered. Each component has a set of possible modes. Behaviour of the system in each mode is described with formulae. Observables are also provided with formulae. Given this model, the task of solving the diagnosis problem consists of finding those modes of the components which are consistent with observations. More formally, the definition of a diagnosis model in the DX community (Hamscher, Console, & de Kleer 1992) is the following :

Definition 3 *A diagnosis model is a triple* $(SD, COMPS, OBS)$ *where :*

- SD *is the system description. It consists of a set of first-order logic formulae which describe the behaviour of the system.*
- $COMPS$ *is a set of constants which represent the components of the system.*
- OBS *is a set of first-order logic formulae which represent the observations given by the sensors.*

The diagnosis problem in the OCSP framework

Like some authors (Williams & Ragno 2003), we argue that a diagnosis problem can be framed as an OCSP (the model) where one must find the n best assignments of mode variables consistent with the constraints describing the system (the task). Decision variables are the mode variables. Solutions are the diagnoses. Non-decision variables are all other variables (including observable and non-observable variables). This has a significant impact because non observable variables may have infinite domains. Considering mode variables as decision variables provides a kind of abstraction that turns an infinite domain CSP into a finite domain CSP. Each mode variable m_{C_i} is attached to one component C_i.

Example The formulation as an OCSP of the diagnosis problem of the classic polybox toy example in the DX community is given in the following.

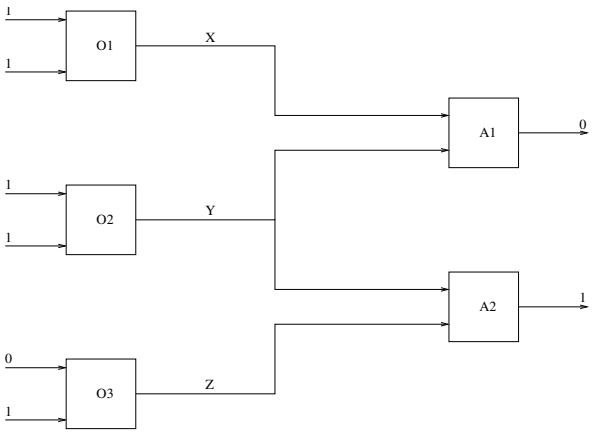


Figure 1: Topology of the boolean polybox

The topology of the polybox is represented in figure 1. O_1, O_2, O_3, A_1, A_2 are the components of the system. O_1, O_2, O_3 are OR gates and A_1, A_2 are AND gates. X, Y, Z are non-observable variables; each of those can take values in $\{0, 1\}$. $m_{O_1}, m_{O_2}, m_{O_3}, m_{A_1}, m_{A_2}$ are mode variables associated to the components; each of those can take values in $\{G, B\}$ (G for Good and B for Bad). A set of constraints link mode variables, observable and non-observable variables. When the observable variable values are provided, one can always instantiate observable variables rather than keeping them in the model. This is why there are no observable variables in the polybox constraint model below.

For sake of clarity we use a constraint language which explicitly describes assignments of values to variables (we could have written the constraints in pure propositional logic). The constraints for the polybox are :

- $(m_{O_1} = G) \implies (X = (1 \vee 1))$
- $(m_{O_2} = G) \implies (Y = (1 \vee 1))$
- $(m_{O_3} = G) \implies (Z = (1 \vee 0))$
- $(m_{A_1} = G) \implies (0 = (X \wedge Y))$
- $(m_{A_2} = G) \implies (1 = (Y \wedge Z))$

These constraints can also be represented in extension because, in this example, the domains of all the involved variables are finite. For instance, the first constraint can be rewritten : $\{(O_1, X), \{(G, 1), (B, 0), (B, 1)\}\}$.

Finally, the diagnosis problem of the polybox in the OCSP framework is :

- decision variables : $\{m_{O_1}, m_{O_2}, m_{O_3}, m_{A_1}, m_{A_2}\}$. Each associated domain is $\{G, B\}$,
- non-decision variables : $\{X, Y, Z\}$. Each associated domain is $\{0, 1\}$,
- constraints are those described above.
- cost function is $g(m_{C_i} = v_i) = \prod_j P_j(m_{C_i})$. Cost represents the candidate probability. The component mode probabilities $P_j(m_{C_i})$ are combined with multiplication because faults on different components are assumed to be independent.

The associated constraint hypergraph is represented in figure 2.

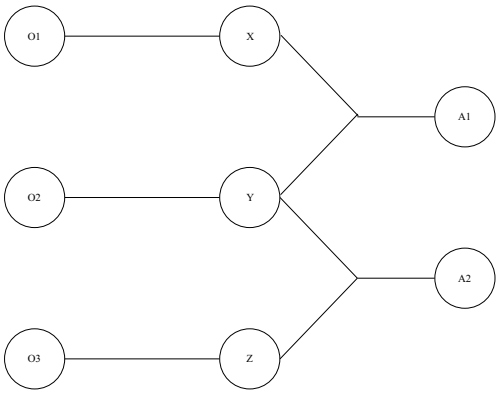


Figure 2: Constraint hypergraph of the boolean polybox

Hierarchical diagnosis and abstraction

Hierarchical diagnosis reasoning

Hierarchical diagnosis reasoning consists of solving a diagnosis problem using an ordered list of diagnosis problems. Generally, this is initiated by a diagnosis problem P_0 and builds up to a more abstract diagnosis problem P_1. Iteratively, one can build an ordered list of n diagnosis problems. Solving these problems is commonly achieved in a top-down fashion. First, the most abstract problem P_{n-1} is solved. Then (or at the same time), knowledge of solutions and non-solutions of P_{n-1} is used (through abstraction) to prune the search space of the problem P_{n-2}. By iterating, one can solve the original diagnosis problem P_0. Results from solving problem P_k can be useful for solving problem P_{k-1} in different ways. It depends on the kind of abstraction which has been used to build problem P_k.

Abstraction concepts

In this section, different views of abstractions are described, each being more or less appropriate to exhibit properties interesting for diagnosis.

Topological view of abstractions (mapping of components) The topological view of abstractions relies on a component-centered representation. Graphically, as shown in figure 3, components are represented by nodes and labeled by their name. The links between components are represented by arcs (direction is chosen according to causality). Each arc is labeled by a value (observable variable) or the name of a non obervable variable. Some abstractions interpret naturally according to the topological view but others cannot be represented in this way. When such a representation exists, topological views of the concrete model and of the abstract model are those giving the best intuition of the abstraction.

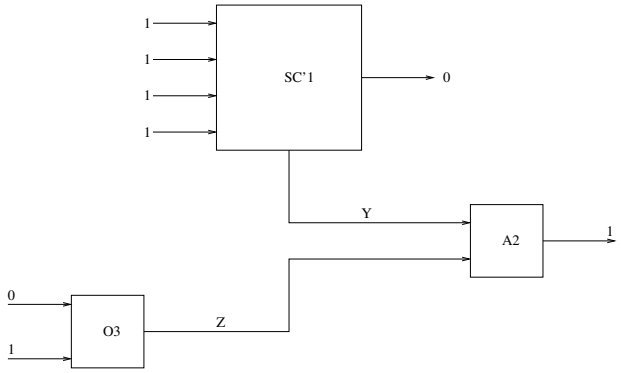


Figure 3: Topology of the abstract boolean polybox.

Example In the polybox example, let us consider a simple structural abstraction (a special case of abstractions) which consists of aggregating for instance the components O_1, O_2 and A_1 in one supercomponent SC'_1. The topology of the abstract polybox is represented in figure 3. But to completely define this abstraction, the mapping between possible values of $(m_{O_1}, m_{O_2}, m_{A_1})$ and $m_{SC'_1}$ must be specified. As an example, a natural mode value mapping is :

- when $(m_{O_1} = G, m_{O_2} = G, m_{A_1} = G)$, then $m_{SC'_1} = G$,
- $m_{SC'_1} = B$ in other cases.

The topological view cannot capture all the abstraction information because the mode value mapping does not explicitely appear.

This mode value mapping is represented in figure 4 and corresponds to the projection on $D(m_{O_1}) \times D(m_{O_2}) \times D(m_{A_1}) \rightarrow D(m_{SC'1})$ of an interpretation mapping as defined in (Nayak & Levy 1995). We can build any value mapping among the 2^8 possible mappings defined on $D(m_{O_1}) \times D(m_{O_2}) \times D(m_{A_1}) \rightarrow D(m_{SC'1})$.

It may be as well the case that mode variables can take more than two values. Some mappings can be more efficient than others, that is discussed in the fourth section. More general kinds of relations than mappings have been deeply described in (Lecoutre *et al.* 2006) but here, just mappings are considered.

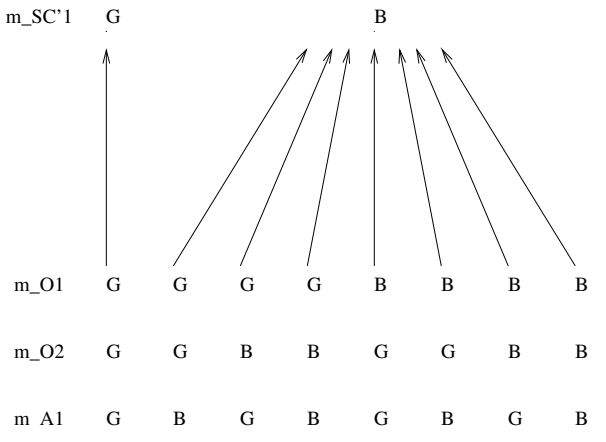


Figure 4: A simple value mapping defined on $D(m_{O_1}) \times D(m_{O_2}) \times D(m_{A_1}) \rightarrow D(m_{SC'1})$.

Limits of the topological view In the last example, one can see that structural abstraction can naturally be defined by two choices :

- aggregation of variables (topological view),
- aggregation of mode value tuples.

However, let us also notice that if we first choose an aggregation of mode value tuples, it implies an aggregation of variables; but the converse does not hold. This remark shows us that criteria for choosing good abstractions, even if it is in the structural abstraction case, can be more easily found by looking directly at the aggregation of mode value tuples (and not at the topological view). This leads us towards a new view of abstractions which corresponds to the search space view.

Search space view (mapping of states) The search space of a diagnosis problem is the set of possible complete assignments to mode variables.

An extended mode value mapping (Lecoutre *et al.* 2006) (Nayak & Levy 1995) associates concrete states to abstract states. This extended mode value mapping can be found extending a mode value mapping to all mode variables. The extended mapping, because its domain spawns on the whole search space, permits to evaluate the number of steps needed by a hierarchical diagnosis algorithm to solve a diagnosis problem. Given that the mode value mapping is generally a surjective mapping (as in the polybox example), the extended mode value mapping is also surjective. Consequently, for each state of the abstract search space, the set of concrete states that correspond to it can be found thanks to the preimage of the extended mapping. The number of elements in this set is called the branching factor of an abstract state. Let us remind to the reader that the preimage of a subset B of the codomain Y under a function f is the subset of the domain X defined by $f^{-1}(B) = \{x \in X | f(x) \in B\}$. The preimage exists even if the mapping is not bijective.

The search space view is difficult to represent by a scheme when the number of components is high because the number of concrete states is exponential in the number of components. In this paper, it is exactly 2^n states for n components. In our polybox example, the 2^5 concrete states are abstracted into 2^3 abstract states by the extended mapping. The latter that we note EM can be found given the mode value mapping M represented in figure 4 in the following manner :

- the domain of EM is $D(m_{O_1}) \times D(m_{O_2}) \times D(m_{O_3}) \times D(m_{A_1}) \times D(m_{A_2}) \rightarrow D(m_{SC'1}) \times D(m_{O_3}) \times D(m_{A_2})$,
- EM maps a concrete state $S = (m_{O_1} = v_1, m_{O_2} = v_2, m_{O_3} = v_3, m_{A_1} = v_4, m_{A_2} = v_5)$ to an abstract state $ES = (m_{SC'1} = M(m_{O_1} = v_1, m_{O_2} = v_2, m_{A_1} = v_4), m_{O_3} = v_3, m_{A_2} = v_5)$.

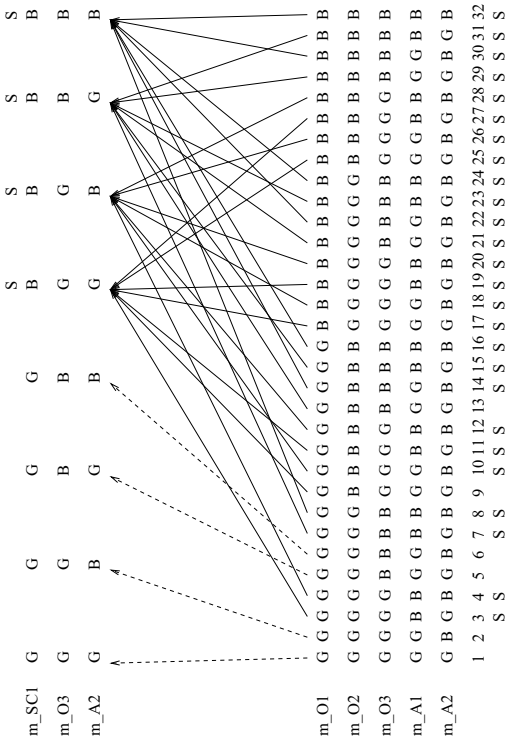


Figure 5: The extended mapping for a structural abstraction of the polybox.

The extended mode value mapping EM is represented in figure 5. Each arrow represents a mapping from a concrete state to an abstract state. The symbol S denotes a state which is solution. One can see that there are 4 solutions to the abstract problem and 26 solutions to the concrete problem.

Constraint view (mapping of constraints) The constraint view of an abstraction is established by the constraint hypergraphs of the concrete and abstract OCSP's.

The constraints of the concrete problem are abstracted into other constraints. In the example of the polybox, constraints 1, 2 and 4 are merged into one constraint : $(m_{SC'_1} = G) \implies ((X = (1 \vee 1)) \wedge (Y = (1 \vee 1)) \wedge (0 = (X \wedge Y)))$. Constraints 3 and 5 are preserved.

The search space view and the constraint view correspond to semantic and syntactic abstractions as defined in (Nayak & Levy 1995), respectively.

Influence of abstractions on computational complexity

The word "best" in "best abstraction" is used for minimal computational temporal complexity, in other words for the minimum number of steps to solve the diagnosis problem. For this analysis, we consider that abstract diagnosis problem(s) are precomputed, so we do not take into account the temporal computational complexity of constructing a hierarchical model of the system to be diagnosed (which is a reasonable hypothesis in the case of on-line state-tracking applications).

The analysis proceeds as follows : firstly, the set of solutions (hence the set of non solutions) is supposed to be known. This permits us to give criteria of "good" abstractions in the space search view; secondly, these criteria are interpreted in the topological and constraint views to determine how they can be used to precompute "good" abstractions that are garantied to be efficient.

Analysis in the search space view

Among all possible abstractions, two general kinds have been identified in the litterature (Nayak & Levy 1995) (Chittaro & Ranon 2004) (Lecoutre *et al.* 2006). :

- Concrete Solution Increasing (CSI),
- Concrete Solution Decreasing (CSD).

Definition 4 (CSI abstraction) *An abstraction is CSI iff for all concrete states which are solutions of the concrete problem, their corresponding abstract state (w.r.t extended mode value mapping) is solution of the abstract problem.*

From this definition, one can trivially deduce the following proposition :

Theorem 1 *Consider a CSI abstraction, then if an abstract state is shown not to be solution, then all its correponding concrete states are not solutions.*

For example, the structural abstraction of the polybox mentioned above is CSI, one can verify it in figure 5.

Definition 5 (CSD abstraction) *An abstraction is CSD iff for all concrete states which are not solutions of the concrete problem, their corresponding abstract state (w.r.t extended mode value mapping) is not solution of the abstract problem.*

From this definition, one can trivially deduce the following proposition :

Theorem 2 *Consider a CSD abstraction, then if an abstract state is shown to be solution, then all its correponding concrete states are solutions.*

A general and simple hierarchical diagnosis algorithm begins from the most abstract problem, generates and tests all the states of a level n in the abstraction hierarchy, then goes down to the level $n-1$, and iteratively, the algorithm finishes to test all the concrete states of the level 0 to give the solutions of the concrete problem.

In this paper, we consider two levels in the abstraction hierarchy but the results can be extended to more than two levels.

Let us recall to the reader that without abstraction, with a simple-minded diagnosis algorithm which just generates and tests states, the temporal computational complexity is $O(n)$ where n is the number of states of the diagnosis problem, i.e. exponential in the number of components.

Consequently, the temporal computational complexity of the hierachical diagnosis algorithm described above depends on two factors :

- the number of abstract states,
- the number of concrete states.

The number of concrete states cannot be changed. An ideal abstraction would consist of two abstract states a and b. a maps all the states which are solutions and b maps all the states which are not. Let us notice that the ideal abstraction we propose is CSI and CSD. So an abstract solution is sufficient to represent all the solutions of the concrete problem; and a non solution abstract state rules out all the non solution concrete states. Consequently, from the CSI and CSD propositions, one can deduce the following proposition :

Theorem 3 *An abstraction that is CSI and CSD reduces the complexity of the diagnosis problem to $O(n')$ where n' is the number of abstract states. The ideal CSI and CSD abstraction has two abstract states.*

But this ideal abstraction cannot be easily built for two reason :

- which states are solutions and which are not is not known in advance,
- constraints associated to the abstract states may not exist.

Refering to the first issue, solutions are supposed to be known (for the analysis) to target which abstractions can easily capture solutions and non solutions. We tackle the second issue using the constraint space view to exhibit abstract constraints which can be easily built.

It appears that the ideal abstraction does not exist in the general case. However one can capture all the solutions and non solutions of a problem using several abstractions of the same problem.

Example

For the polybox example, there are 32 states and, among them, 26 solutions. The diagnosis community is used to represent the 26 solutions by 3 minimal diagnoses : $\{O_1\}, \{A_1\}, \{O_2, A_2\}$. These minimal diagnoses represent 16, 13 and 8 solution states, respectively. Some states are represented by several minimal diagnoses. Non solution states can be captured by minimal conflicts. Minimal conflicts $\{O_1, O_2, A_1\}$ and $\{O_1, A_1, A_2\}$ each captures 4 non solutions.

The ideal abstraction of the polybox example maps the 26 solutions to one abstract state a and the 6 non solutions to another abstract state b. But with such a partition of the search space, building abstract constraints means solving diagnosis problem. However, given that we want to build hierarchical model for all possible set of inputs, this approach is not feasible. One may however notice that using abstract states corresponding to minimal potential conflicts (Cordier

et al. 2004), we can build a relatively efficient hierarchical model for all sets of inputs. All non solutions are captured and ruled out in an efficient way. This idea is developped in the next subection.

Using constraints to find "best" abstractions

We can relax the ideal abstraction requirement using several "two abstract states-based" abstractions to capture the whole search space. One needs, for each abstraction, one abstract state which can represent the highest number of non solutions and another abstract state to represent the highest number of solutions.

To choose these states, the topological and constraint views of abstractions are used. Choosing abstractions in the search space view may lead to non existant abstract CSPs, i.e. for which one cannot find the corresponding variables and constraints.

To represent a maximal number of states, one has two main choices :

- aggregating variables and mode values as described in the topological section,
- removing variables and/or values.

In the constraints view, aggregating variables corresponds to merging constraints and removing variables corresponds to removing constraints. The first option exactly corresponds to structural abstraction and one can see in the polybox example that with natural mappings, abstraction is CSI but does not permit to rule out a lot of non solution states in one test. For the second option, removing n variables permits one to represent 2^n states. So, one has to remove the highest number of variables while keeping the constraints decidable. Removing variables means removing their associated constraints. Consequently, one must find the smallest set of constraints which remain decidable. These sets are hence just overdeterminated, i.e. they involve n equations for $n-1$ unknowns, and they are well-known in the diagnosis field as corresponding to minimal potential conflicts (Pulido & Alonso 2002) (Cordier *et al.* 2004).
When the test on a given OBS of one such just overdeterminated constraint set does not pass, the conjunction of assignments to G of the involved mode variables is a minimal conflict. The case in which at least one variable is assigned to B does not help for finding solutions since the constraints are always satisfied. It is easy to show that when using a minimal potential conflict to construct an abstraction, this abstraction is CSI but not CSD. It is also known that all minimal potential conflicts capture all concrete non solutions. So checking all the abstract states built this way garanties to rule out all the non solution states. However, one abstraction per minimal potential conflict is necessary since one concrete state may correspond to more than one minimal potential conflict. Indeed, we have restricted our analysis to mappings between concrete states and abstract states. With general relations, we conjecture that one abstraction can contain all minimal potential conflicts but in this case, one concrete state may have more than one image.

Conclusion and future work

Hierarchical diagnosis efficiency strongly depends on the choice of the abstractions used to build a hierarchical model of the system. For analysing the influence of this choice, three complementary views of abstractions have been defined : the topological view, the search space view and the constraint view. In the case in which abstractions are built removing variables or contraints, we argue that most efficient abstractions are obtained from minimal potential conflicts because they rule out all non solution states.
But finding efficient abstractions has a cost. For instance, determining minimal potential conflicts is known to be of exponential complexity. So our future work will focus on defining an efficiency measure for an abstraction that takes in account not only the computational gain on the hierarchical model but also the cost of finding the model. In particular, we will pay much attention to structural abstraction because it is generally cheap; and we will consider building a bridge between structural abstractions and other types of abstractions (like conflict-based) that all together may boost the diagnosis resolution process efficiency.

References

Chittaro, L., and Ranon, R. 2004. Hierarchical model-based diagnosis based on structural abstraction. *Artif. Intell.* 155(1-2):147–182.

Cordier, M. O.; Dague, P.; Levy, F.; Montmain, J.; Staroswiecki, M.; and Travé-Massuyès, L. 2004. Conflicts versus analytical redundancy relations : a comparative analysis of the model-based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man and Cybernetics.* 34(5):2163–2177.

Hamscher, W.; Console, L.; and de Kleer, J. 1992. *Readings in model-based diagnosis*. Morgan Kaufmann.

Lecoutre, C.; Merchez, S.; Boussemart, F.; and Grégoire, E. 2006. A csp abstraction framework. *Revue d'intelligence artificielle* 20(1):31–62.

Nayak, P., and Levy, A. 1995. A semantic theory of abstractions. In Mellish, C., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 196–203. San Francisco: Morgan Kaufmann.

Pulido, B., and Alonso, C. 2002. Possible conflicts, arrs, and conflicts. *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis (DX 02)*.

Torta, G., and Torasso, P. 2006. Qualitative domain abstractions for time-varying systems: an approach based on reusable abstraction fragments. In *Proc 17th Workshop on Principle of Diagnosis (DX 06)*.

Williams, B. C., and Ragno, R. J. 2003. Conflict-directed a* and its role in model-based embedded systems. *Journal of Discrete Applied Math*.

Diagnosability Analysis for Web Services with Constraint-based Models

Xavier Pucel[(a)], Stefano Bocconi[(b)], Claudia Picardi[(b)], Daniele Theseider Dupré[(c)], Louise Travé-Massuyès[(a)]

(a) LAAS-CNRS, Université de Toulouse, 7, av. du Colonel Roche, 31077 Toulouse France.
(b) Università di Torino, Dipartimento di Informatica, Corso Svizzera 185, 10149 Torino Italy.
(c) Università del Piemonte Orientale, Dipartimento di Informatica, Via Bellini 25/g, 15100 Alessandria Italy.
{*xpucel,louise*}*@laas.fr,* {*Stefano.Bocconi,Claudia.Picardi*}*@di.unito.it, dtd@mfn.unipmn.it*

Abstract

In this paper we deal with the problem of model-based diagnosability analysis for Web Services. The goal of diagnosability analysis is to determine whether the information one can observe during service execution is sufficient to precisely locate (by means of diagnostic reasoning) the source of the problem. The major difficulty in the context of Web Services is that models are distributed and no single entity has a global view of the complete model. In the paper we propose an approach that determines diagnosability for the decentralized diagnostic framework, described in (Ardissono *et al.* 2005), based on a Supervisor coordinating several Local Diagnosers. We also show that diagnosability analysis can be performed without requiring the Local Diagnosers different operations than those needed for diagnosis. The diagnosability of each fault mode is first analyzed independently of the occurrence of other faults, then the results are used to analyze some combinations of modes explicitly, and leave the analysis of other combinations implicit, to avoid an exhaustive check.

Introduction

Diagnosability analysis is a requisite for analyzing and designing systems, however this task is particularly difficult when addressing complex systems such as Web Services. Decomposition has been recognized as an important leverage to manage architectural complexity of such systems. On the one hand, some approaches like (Pencolé & Cordier 2005) take advantage of the decomposition of the system to decrease the amount of computation needed for diagnosability analysis. On the other hand, new problems have to be solved when independently designed subsystems are assembled, for example in the case of a business process involving several distinct partners. Each subsystem may come with its own diagnosis and diagnosability analysis software whose details should not be disclosed to the external world, even to business partners.

The diagnosis approach developed in (Ardissono *et al.* 2005) allows one to deal with this privacy issue, by using Local Diagnosers developed by each partner and that only disclose information about the subsystem's interfaces. Local Diagnosers communicate with a Supervisor via a common interface. This paper presents a diagnosability analysis approach that is compatible with the context described above, taking advantage of the existing diagnosis architecture in order to solve distribution and privacy issues.

The approach we propose performs diagnosability analysis as a two-phase process. In the first phase, it builds the analogous of a fault table, that is a table matching faults to observables. This table is built in a decentralized way, and its peculiarity is that the entries are not just combinations of faults, but rather *partial fault modes* where the presence of some fault is left unspecified. As we will see, this simplifies the second phase, which consists of comparing the different entries of the table in order to assess diagnosability-related properties. In this paper we will assume that at the end of the first phase the "fault table" is completely stored in the Supervisor, which can then perform the second phase on its own. This requires that the Supervisor sees something more of the local models (namely, observable variables) with respect to the approach described in (Ardissono *et al.* 2005).

This paper is organized as follows: first, we present a brief description of the diagnosis approach in (Ardissono *et al.* 2005). Then, we introduce some general notions related to diagnosability, and we use them to define our diagnosability analysis algorithm and the properties on which it relies. We then present a complete example and discuss related work.

The diagnosis approach

This paper is based on the decentralized diagnostic approach in (Ardissono *et al.* 2005; Console, Picardi, & Theseider Dupré 2007) which was developed for Web Services. In this section we recall the contributions that are relevant for the diagnosability analysis discussed later.

The overall architecture is based on several **Local Diagnosers** $A_1, \ldots, A_n$ which cooperate with a **supervisor** $\boldsymbol{D}$. Each Local Diagnoser A_i is responsible for a Web Service W_i (or a set of Web Services), while $\boldsymbol{D}$ puts together information from Local Diagnosers and selects which Local Diagnosers to question further in order to diagnose problems.

Each Local Diagnoser possesses a *model* of the Web Service; the approach makes the following assumptions on models:

- Each model is given as a set of constraints over finite-domain variables.
- A WS model is based on its workflow, which can be seen as a partially-ordered set of *activities*. Activities are the minimal diagnosable unit: diagnosis aims at discovering

which of the activities in the workflow first caused the observed problem.

- For each activity there is a distinguished *mode* variable that expresses which behaviour mode (ok or faulty) the activity is in. In this paper we will consider, for the sake of simplicity, that there is only one fault mode named *ab*.
- Interactions with other services are represented by means of "shared" variables (where "shared" means that each model has its own variable, and an implicit equality constraint exists between the two). These variables are called *interface* variables.

Work of Local Diagnosers and supervisor is based on *partial assignments* to model variables (in particular, to mode variables and to interface variables). A partial assignment corresponds to some hypothesis of behaviour of part of the system; variables that are not constrained by the hypothesis should remain unassigned in order to keep the hypothesis as general as possible. More precisely, since details of the models should not necessarily be disclosed within a distributed environment, variables which should remain unassigned are those that the partial assignment and the model do not constrain more than the model alone.

In the diagnostic approach such a least commitment is used to avoid exploring parts of the model that need not be explored. Since every time that a variable in a model is constrained the corresponding Local Diagnoser is invoked, leaving variables unassigned when not relevant to the diagnostic process prevents the Supervisor from invoking Local Diagnosers unnecessarily.

The operation performed by a Local Diagnoser in order to explain an abnormal behaviour is called the **Extend** operation, each output partial assignment being an *extension* of an input partial assignment, and is used to propagate hypotheses across a subsystem, either to find local causes of an abnormal situation or to explore the consequences of an hypothesis.

In order to characterize this operation, we introduce some definitions, reformulated from (Ardissono *et al.* 2005; Console, Picardi, & Theseider Dupré 2007).

Definition 1. *Let M be a model and let γ be a partial assignment. Moreover, let M_γ denote the model obtained by constraining M with γ. We say that γ is* admissible *with respect to M if:*

i. γ is consistent with M;

ii. the restriction of M_γ to variables not *assigned by γ is equivalent to the restriction of M itself to the same variables: $M_\gamma|_{\text{VAR}(M)\setminus \boldsymbol{Dom}(\gamma)} \equiv M|_{\text{VAR}(M)\setminus \boldsymbol{Dom}(\gamma)}$.*

Admissibility captures the requirement mentioned above: unassigned variables are not constrained by the assignment more than the model already does.

For a given input assignment α (corresponding to observations in the diagnosis case), a compact representation of the set of scenarios corresponding to α would be the set of *minimal* admissible extensions of α, where "minimal" means "as general as possible". However, as discussed in (Console, Picardi, & Theseider Dupré 2007), in general this cannot be done without Local Diagnosers sharing more information with each other about their local models (something that is not desirable in a distributed environment). Then the weaker notion of *complete set of admissible extensions* is used.

Definition 2. *Let M be a model and let γ be a partial assignment. A set E of admissible assignments extending γ is* complete *if every total assignment consistent with M and γ is an extension of some $\delta \in E$.*

In the diagnostic approach, for each assignment α in input, a local diagnoser computes a complete set of extensions (wrt the local model M_i) for $\alpha \wedge \omega$ where ω are local observations that are performed by the Local Diagnoser and can, of course, discard some hypotheses.

The overall diagnostic process starts from an abnormal observation in service i, and the Local Diagnoser of service i is asked to compute a complete set of admissible extensions for the assignment corresponding to the observation.

If an extension computed by the Local Diagnoser i assigns a value to an interface variable shared by service j, then the Local Diagnoser for service j is invoked by the supervisor to further extend the assignment, taking into account the model of service j, and observations (if any) of service j, which may discard the hypothesis. At the end of the supervisor loop, a complete set of admissible extensions for the observations obtained in the process is computed.

In the approach described in the following sections, the same decentralized algorithm is used to predict observable consequences of some faults, *independently* of the presence of other faults.

Example

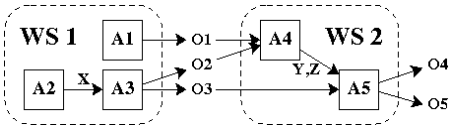
The notions described in this article are illustrated by a small example. The way in which we model WSs in this example is taken from (Ardissono *et al.* 2005), although we give here a simplified version.

In the example, there are two Web Services WS_1 and WS_2, made of two and three activities respectively. Each activity model has a *mode* variable, representing the correctness status of the activity, and one or more *data variables* that represent the correctness status of input and/or output data.

Each activity is modelled as a constraint expressing how the correctness of output data depends on the correctness of the inputs and of the activity itself. For example, activity A_3 has one inputs and two outputs. The model states that if either the activity or the input is incorrect, then both the outputs are incorrect as well. However, if *both* the activity and the input are incorrect, then only output o_3 is incorrect (in o_2 the two abnormalities mask each other).

The composed model is typically obtained by adding equality constraints between connected variables; in order to keep the example as simple as possible we directly used the same variable name rather than adding an equality constraint.

We can now use this example to illustrate the notions introduced in the previous section. Let us consider the partial assignment $\alpha \equiv (m_4 = ok)$. With respect to the model of



Communication model

m_1	o_1
ok	*ok*
ab	*ab*

Activity A_1

m_2	x
ok	*ok*
ab	*ab*

Activity A_2

m_3	x	o_2	o_3
ok	*ok*	*ok*	*ok*
ok	*ab*	*ab*	*ab*
ab	*ok*	*ab*	*ab*
ab	*ab*	*ok*	*ab*

Activity A_3

m_4	o_1	o_2	y	z
ok	*ok*	*ok*	*ok*	*ok*
ok	*ab*	*ok*	*ok*	*ab*
ab	*ok*	*ok*	*ab*	*ok*
ab	*ab*	*ok*	*ab*	*ab*
ok	*ok*	*ab*	*ok*	*ok*
ok	*ab*	*ab*	*ok*	*ab*
ab	*ok*	*ab*	*ab*	*ok*
ab	*ab*	*ab*	*ab*	*ab*

Activity A_4

m_5	y	z	o_3	o_4	o_5
ok	*ok*	*ok*	*ok*	*ok*	*ok*
ok	*ok*	*ab*	*ok*	*ok*	*ok*
ok	*ab*	*ok*	*ok*	*ab*	*ab*
ok	*ab*	*ab*	*ok*	*ok*	*ab*
ab	*ok*	*ok*	*ok*	*ok*	*ab*
ab	*ok*	*ab*	*ok*	*ab*	*ok*
ab	*ab*	*ok*	*ok*	*ab*	*ab*
ab	*ab*	*ab*	*ok*	*ab*	*ab*
ok	*ok*	*ok*	*ab*	*ok*	*ok*
ok	*ok*	*ab*	*ab*	*ok*	*ok*
ok	*ab*	*ok*	*ab*	*ab*	*ab*
ok	*ab*	*ab*	*ab*	*ok*	*ab*
ab	*ok*	*ok*	*ab*	*ok*	*ab*
ab	*ok*	*ab*	*ab*	*ab*	*ok*
ab	*ab*	*ok*	*ab*	*ab*	*ab*
ab	*ab*	*ab*	*ab*	*ab*	*ab*

Activity A_5

Figure 1: Illustrative example: two Web Services composed of five activities.

A_4, this assignment is not admissible, since it does not meet requirement *(ii)* for admissibility.

On the other hand, if we consider $\beta \equiv (m_4 = ok \wedge y = ab)$ (which is an extension of α), we see that β is admissible. Figure 2 shows the table for $M|_{\text{VAR}(M)\setminus\textbf{Dom}(\beta)}$, which is equivalent to $M_\beta|_{\text{VAR}(M)\setminus\textbf{Dom}(\beta)}$.

It is also worth noting that the singleton $\{\beta\}$ is a *complete set* of admissible extensions for α. In fact, all tuples in M_α are extensions of β.

Diagnosability Analysis

This section presents a diagnosability analysis algorithm in relation with the diagnosis algorithm described in the previous section. Diagnosability analysis is performed by determining which system behaviours are pairwise discriminable.

Preliminary concepts

First of all, let us define formally what we mean by *fault* and *fault mode*. A *fault* is characterized by an abnormal behavior of one of the atomic activities involved in the WS. An activity is said to be faulty when a fault has occurred in it. In the example's model, a fault is an assignment to *ab* to a single mode variable.

A *fault mode* is relative to the whole system: it describes the state (among normal and faulty states) of all the activities of the composite WS. It is represented by an assignment to *all* mode variables. Diagnosis consists in deciding in which fault mode the system is, by assessing which faults have occurred and which have not. In this paper's example, the unique normal mode is characterized by an assignment of only *ok* values.

o_1	o_2	z
ok	*ok*	*ok*
ab	*ok*	*ab*
ok	*ab*	*ok*
ab	*ab*	*ab*

Figure 2: $M|_{\text{VAR}(M)\setminus\textbf{Dom}(\beta)}$ with M model of A_4 and $\beta \equiv (m_4 = ok \wedge y = ab)$.

Our approach is based on the analysis of *partial fault modes*, defined as follows together with their properties.

Definition 3 (Partial fault mode)**.** *A partial fault mode is defined by assigning a value to* some *of the system mode variables. A partial fault mode in which* all *mode variables are assigned is a* fault mode*.*

The *domain* of a partial fault mode is the set containing the mode variables it assigns. The *rank* of a partial fault mode is the cardinality of its domain.

Two partial fault modes are *alternative* when they have the same domain, but are inconsistent with each other. A partial fault mode pfm_1 *refines* another pfm_2 if every assigned variable in pfm_2 is assigned with the same value in pfm_1 and pfm_1 has a rank strictly greater than pfm_2 (pfm_1 is an extension of pfm_2). For example, $m_1 = ok \wedge m_2 = ok$ and $m_1 = ok \wedge m_2 = ab$ are alternative partial fault modes, while $m_1 = ab \wedge m_2 = ok$ refines $m_1 = ab$.

Definition 4 (Signature)**.** *Let M_{pfm} be the model obtained by constraining M with a partial fault mode pfm, the* signature *of pfm is the restriction*[1] *of M_{pfm} to observable variables.*

This definition is consistent with the signature definition given in (Cordier, Travé-Massuyès, & Pucel 2006) which applies only to fault modes: the signature of a partial fault mode *pfm* is the union of the signatures of all fault modes refining it.

Definition 5 (Discriminability)**.** *Two partial fault modes are* discriminable *if and only if their signatures are inconsistent, or, when considered as a set of tuples, disjoint.*[2]

[1] Other approaches use the term projection for the same concept.

[2] When applied to fault modes, this notion is also called *strong discriminability* (Travé-Massuyès, Escobet, & Olive 2006; Rozé & Cordier 2002) or *necessary discriminability* (Struss & Dressler

The following property is a straightforward consequence of the previous definitions:

Property 1. *Two partial fault modes pfm_1 and pfm_2 are discriminable if and only if each refinement of pfm_1 is discriminable from each refinement of pfm_2.*

Diagnosability definitions based on dicriminability generally apply to fault modes, however property 1 allows us to state the notion of diagnosability in a more generic way, fault modes being particular cases of partial fault modes:

Definition 6 (Diagnosability)**.** *A partial fault mode is* diagnosable *if and only if it is discriminable from all its alternatives.*
A system is diagnosable if and only if all its partial fault modes are diagnosable.

Detectability can also be defined as follows:

Definition 7 (Detectability)**.** *A partial fault mode is* detectable *if and only if:*

1. *it assigns ab to at least one mode variable (it is not consistent with the all-ok mode).*
2. *it is discriminable from its alternative assigning only ok values.*

This definition implies fault mode detectability as defined in (Cordier *et al.* 2004), thanks to property 1. This property can be enriched with the previous definitions:

Property 1'. *A partial fault mode is diagnosable (resp. detectable) if and only if all its refinements are diagnosable (resp. detectable).*

Approach

Our approach to diagnosability analysis has the goal of finding which pairs of fault modes are discriminable. As we will shortly see, in the decentralized case it is convenient to perform this analysis for pairs of *partial* fault modes rather than total ones.

The supervised approach to diagnosis in (Console, Picardi, & Theseider Dupré 2007), briefly described in the previous section, allows to propagate hypotheses (in that case, diagnostic hypotheses) throughout distributed models, provided that the model owners are willing to share with the Supervisor the values of their interface variables.

If we assume that the Supervisor is entitled also to observable variables, it is then possible to use this approach to propagate each fault mode and compute the set of observable variables corresponding to it.

The Supervisor then compares the sets of observable variables and determines which pairs of fault modes are indeed discriminable.

This approach has however one major limitation: in order to be able to propagate hypotheses, the Supervisor must store the information on the possible values of those interface variables that are dependent from the hypothesis itself. When constraining all mode variables, as it happens when the hypothesis is a fault mode, all interface variables are dependent on it. This means that the amount of information the Supervisor needs to store can easily explode.

2003) in the literature.

In the case of diagnosis, the problem was dealt with by considering as diagnostic candidates *partial* fault modes, that assign a value only to those variables on which the observed misbehavior could depend on. In this way, the propagation of information is restricted only to those variables that are concretely related to the hypothesis. This does not solve the problem in the worst case (when everything depends on everything - as it is discussed in (Darwiche 1998) tractability of diagnosis depends on the degree of interconnection between its components) but it makes it tractable in many practical cases, as for example the case of Web Services.

Our goal is now to show that, by adopting a similar approach based on partial fault modes, and by exploiting some properties of complete sets of admissible extensions, we can perform supervised diagnosability analysis and :

- control, when possible, the explosion in the size of information the Supervisor needs to store;
- obtain a more general discriminability analysis, in terms of partial fault modes rather than total ones.

The approach we propose consists of the following steps:

(1) The Supervisor computes a complete set of admissible extensions of all partial fault modes of rank 1. It does so by exploiting the diagnosis approach. However, in this case the Supervisor maintains in the hypothesis table also observable variables (while in the original diagnosis approach the hypothesis table contained only mode and interface variables).

(2) After the first step is completed, the Supervisor does not need the Local Diagnosers anymore. It can thus discard from the hypothesis table all the interface variables, keeping only mode and observable variables. Notice that the rows of this table still represent a complete set of admissible extensions.

(3) The Supervisor then performs discriminability analysis rank by rank. A discriminability analysis of rank k consists in comparing two alternative partial fault modes of rank k by exploiting their complete sets of admissible extensions. As we will see, we will equip the Supervisor with means for pruning the search space, avoiding rank k analyses whose uselessness is already apparent at lower ranks.

The next section states the properties of complete sets of admissible extensions that allow this type of analysis. We then detail the approach on the basis of those properties.

Properties

The first property shows that Local Diagnosers are needed only in computing complete sets of admissible extensions for partial fault modes of rank 1.

Property 2. *Let pfm_1 and pfm_2 be two consistent partial fault modes, and let us assume to have a complete set* **Ext**(pfm_i) *of admissible extensions for each of them. Then the set*

$$\{\alpha_1 \wedge \alpha_2 \mid \alpha_1 \in \textbf{Ext}(pfm_1), \alpha_2 \in \textbf{Ext}(pfm_2), \alpha_1 \textit{ consistent with } \alpha_2\}$$

is a complete set of admissible extensions for $pfm_1 \wedge pfm_2$.

This allows us to compute complete sets of admissible extensions for rank k partial fault modes by combining the sets obtained at rank 1.

The next property is the one on which the whole approach rests: it states that discriminability of two partial fault modes can be assessed by comparing their complete sets of admissible extensions.

Property 3. *Let pfm_1 and pfm_2 be two alternative partial fault modes, and let us assume to have a complete set* **Ext**(pfm_i) *of admissible extensions for each of them. Then pfm_1 and pfm_2 are discriminable if and only if for all $\alpha_1 \in$* **Ext**$(pfm_1), \alpha_2 \in$ **Ext**(pfm_2)*, their restrictions α'_1 and α'_2 to observable variables are not consistent (i.e. there is at least one variable for which they assign different values).*

The next property is the most complex. It enables us to have an early detection of hopelessly non-discriminable pairs, so to avoid refining them at higher ranks.

Property 4. *Let pfm_1 and pfm_2 be two alternative not discriminable partial fault modes. Let D denote their (coincident) domain and* **Ext**(pfm_1)*,* **Ext**(pfm_2) *their complete sets of admissible extensions.*
Let m be a mode variable with **Dom**$(m) \cap D = \emptyset$*.*
If for every $\alpha_1 \in$ **Ext**$(pfm_1), \alpha_2 \in$ **Ext**(pfm_2)*, consistent with each other when restricted to observable variables, it holds that* **Dom**$(\alpha_1) \cap$ **Dom**$(m) = \emptyset$ *and* **Dom**$(\alpha_2) \cap$ **Dom**$(m) = \emptyset$*, then the combinations $\{(pfm_1 \wedge (m = v_1), pfm_2 \wedge (m = v_2)) \mid v_{1,2} \in \{ok, ab\}\}$ are non-discriminable as well.*

Intuitively, this tells us that whenever at rank k a pair of partial fault modes pfm'_1, pfm'_2 is non discriminable, and the extensions of pfm'_1, pfm'_2 do not mention a given additional mode variable m, then we can avoid to further refine this pair with m. In fact, in this case $pfm''_1 \equiv (m = v_1), pfm''_2 \equiv (m = v_2)$ and therefore :

- if pfm''_1 and pfm''_2 are non-discriminable (in particular if $v_1 = v_2$), then due to property 4 the refined assignments will be non-discriminable as well;
- if pfm''_1 and pfm''_2 are discriminable, then the refined assignments will be trivially discriminable and their discriminability is discovered during rank 1 analysis.

Algorithm

In this section we show how step **(3)** can be detailed taking into account the search space pruning allowed by the properties discussed before.
The main structure of step **(3)** is as follows:

OUTPUT $= \emptyset$; DISC$_1 = \emptyset$; $k = 1$;
TODO$_1 = \{(m = ok, m = ab) \mid m$ mode variable$\}$;
while ($k \leq$ *maxrank* $\wedge$ TODO$_k \neq \emptyset$)
  for each pair $(pfm_1, pfm_2) \in$ TODO$_k$
    EXT$_1$ = **Extend**(pfm_1); EXT$_2$ = **Extend**(pfm_2);
    AddDisc(DISC$_k$, EXT$_1$, EXT$_2$, pfm_1, pfm_2);
    AddToDo(TODO$_{k+1}$, EXT$_1$, EXT$_2$, pfm_1, pfm_2);
  OUTPUT = OUTPUT $\cup$ DISC$_k$;
  DISC$_{k+1}$ = **Update**(DISC$_k$, $k+1$);
  TODO$_{k+1}$ = TODO$_{k+1} \setminus$ DISC$_{k+1}$;
return **Expand**(OUTPUT, k);

The algorithm has a main loop that proceeds rank by rank until either the maximum rank has been reached, or all the pairs of partial fault modes at higher ranks need not be analyzed thanks to the properties. At the end of the algorithm OUTPUT will contain the set of all pairs of discriminable alternative partial fault modes of all ranks (including those that the algorithm did not explicitly analyze).

At iteration k, TODO$_k$ contains the set of pairs of alternative partial fault modes of rank k that should be analyzed for discriminability. The goal of iteration k is to find discriminable pairs of rank k, adding them to the output set, and to prepare the pairs that should be analyzed during iteration $k+1$. For these reasons it computes two sets: DISC$_k$ (discriminable pairs of rank k) and TODO$_{k+1}$ (pairs to be analyzed in the next iteration).

For each element of a pair $(pfm_1, pfm_2) \in$ TODO$_k$ we first of all need to have a complete set of admissible extensions. These sets either have already been computed in step **(1)** (in this case **Extend** simply retrieves them from some lookup table) or are obtained by **Extend** composing rank 1 results (see property 2). Then the pair is examined and:

- DISC$_k$ is updated if the pair is found discriminable. The function **AddDisc** simply checks the discriminability of a pair (using property 3) and possibly adds it to the DISC$_k$ set.
- TODO$_{k+1}$ is updated to include all the pair refinements at rank $k+1$ that should be analyzed at next iteration, because they do not meet the conditions of property 4. Such refinements are computed by the **AddToDo** function as follows. Let us denote by D the common domain of pfm_1 and pfm_2. Then for each pair of partial assignments $\alpha_1 \in$ EXT$_1, \alpha_2 \in$ EXT$_2$ consistent with each other on observable variables, if α_1 or α_2 assigns a value to a mode variable $m \notin D$, then the following set of refinements is added to TODO$_{k+1}$:
  $\{(pfm_1 \wedge (m = v_1), pfm_2 \wedge (m = v_2)) \mid v_{1,2} \in \{ok, ab\}\}$

Finally, the set DISC$_{k+1}$, representing refinements of rank $k+1$ of discriminable pairs, is computed from DISC$_k$ and subtracted from TODO$_{k+1}$, since these pairs are trivially discriminable. DISC$_k$ is added to the final output set.

Since the analysis, thanks to search space pruning, could end before reaching the maximum rank, the final output set is obtained by expanding all the discriminable pairs found during the loop to higher ranks.

Example

In this example we apply the algorithm defined in the previous section to the Web Services described in figure 1. In a slight abuse of notation, "m_i is diagnosable" is used to express that $m_i = ok$ and $m_i = ab$ are discriminable, hence diagnosable.

Rank 1 At step one the Supervisor computes a complete set of admissible extensions of all partial mode assignments of rank 1 (see figure 3). The information gathered at this stage suffices for the rest of the analysis, and the Supervisor does not need to invoke the Local Diagnosers anymore. As we said earlier, only observable and mode variables are kept,

pfm	mode vars					observable vars				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$m_1 = ok$	*ok*	$*$	$*$	*ok*	*ok*	*ok*	$*$	$*$	*ok*	*ok*
	ok	$*$	$*$	*ok*	*ab*	*ok*	$*$	$*$	*ok*	*ab*
	ok	$*$	$*$	*ab*	$*$	*ok*	$*$	$*$	*ab*	*ab*
$m_1 = ab$	*ab*	$*$	$*$	*ok*	*ok*	*ab*	$*$	$*$	*ok*	*ok*
	ab	$*$	$*$	*ok*	*ab*	*ab*	$*$	$*$	*ab*	*ok*
	ab	$*$	$*$	*ab*	*ok*	*ab*	$*$	$*$	*ok*	*ab*
	ab	$*$	$*$	*ab*	*ab*	*ab*	$*$	$*$	*ab*	*ab*
$m_2 = ok$	$*$	*ok*	*ok*	$*$	$*$	$*$	*ok*	*ok*	$*$	$*$
	$*$	*ok*	*ab*	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$
$m_2 = ab$	$*$	*ab*	*ok*	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$
	$*$	*ab*	*ab*	$*$	$*$	$*$	*ok*	*ab*	$*$	$*$
$m_3 = ok$	$*$	*ok*	*ok*	$*$	$*$	$*$	*ok*	*ok*	$*$	$*$
	$*$	*ab*	*ok*	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$

pfm	mode vars					observable vars				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$m_3 = ab$	$*$	*ok*	*ab*	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$
	$*$	*ab*	*ab*	$*$	$*$	$*$	*ok*	*ab*	$*$	$*$
$m_4 = ok$	$*$	$*$	$*$	*ok*	*ok*	$*$	$*$	$*$	*ok*	*ok*
	ok	$*$	$*$	*ok*	*ab*	*ok*	$*$	$*$	*ok*	*ab*
	ab	$*$	$*$	*ok*	*ab*	*ab*	$*$	$*$	*ab*	*ok*
$m_4 = ab$	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$	$*$	*ab*	*ab*
	ok	$*$	$*$	*ab*	$*$	*ok*	$*$	$*$	*ab*	*ab*
	ab	$*$	$*$	*ab*	*ok*	*ab*	$*$	$*$	*ok*	*ab*
$m_5 = ok$	$*$	$*$	$*$	*ok*	*ok*	$*$	$*$	$*$	*ok*	*ok*
	ok	$*$	$*$	*ab*	*ok*	*ok*	$*$	$*$	*ab*	*ab*
	ab	$*$	$*$	*ab*	*ok*	*ab*	$*$	$*$	*ok*	*ab*
$m_5 = ab$	$*$	$*$	$*$	*ab*	*ab*	$*$	$*$	$*$	*ab*	*ab*
	ok	$*$	$*$	*ok*	*ab*	*ok*	$*$	$*$	*ok*	*ab*
	ab	$*$	$*$	*ok*	*ab*	*ab*	$*$	$*$	*ab*	*ok*

Figure 3: The admissible extensions of all partial fault modes of rank 1 (restricted to mode and observable variables).

while interface variables are discarded (none in this case, since all interface variables are observable, see figure 1).

At this point the Supervisor starts to perform the diagnosability analysis from rank 1. Looking at the observable variables, we see m_1 and m_4 are diagnosable, since the extensions of alternative pairs restricted to observable variables are not consistent. Thus, the pair of assignments $(m_1 = ok, m_1 = ab)$ (resp. $(m_4 = ok, m_4 = ab)$) is inserted in the DISC$_1$ set. As a consequence, each refinement of $m_1 = ok$ (resp. $m_4 = ok$) is discriminable from each refinement of $m_1 = ab$ (resp. $m_4 = ab$). These pairs of refinements are inserted in the DISC$_2$ set for further use.

Continuing with rank 1 analysis, the algorithm finds that:

- m_2 is not diagnosable (considering only restrictions to observable variables, the 2nd and 3rd extensions of $m_2 = ok$ are consistent with the 1st extension of $m_2 = ab$)
- pairs of partial fault modes in the domain $\{m_2, m_3\}$ need to be checked, since m_3 is present in the extensions of m_2 (property 4).

Therefore, the algorithm inserts in the TODO$_2$ all pairs of partial fault modes in the domain $\{m_2, m_3\}$:

$(m_2 = ok \wedge m_3 = ok, m_2 = ok \wedge m_3 = ab)$
$(m_2 = ok \wedge m_3 = ok, m_2 = ab \wedge m_3 = ok)$
$(m_2 = ok \wedge m_3 = ok, m_2 = ab \wedge m_3 = ab)$
$(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ok)$
$(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ab)$
$(m_2 = ab \wedge m_3 = ok, m_2 = ab \wedge m_3 = ab)$

Analyzing the last fault mode variable, m_5, the algorithm determines that it is also non diagnosable (considering only restrictions to observable variables, the 2nd extension of $m_5 = ok$ is consistent with the 1st extension of $m_5 = ab$), and that it needs to check at rank 2 the pairs of partial fault modes with a domain included in $\{m_1, m_4, m_5\}$.

Some of those pairs are contained into the DISC$_2$ set, and do not need to be checked, since m_1 and m_4 are diagnosable. Therefore only the following four combinations are inserted in TODO$_2$:

$(m_5 = ok \wedge m_1 = ok, m_5 = ab \wedge m_1 = ok)$
$(m_5 = ok \wedge m_1 = ab, m_5 = ab \wedge m_1 = ab)$
$(m_5 = ok \wedge m_4 = ok, m_5 = ab \wedge m_4 = ok)$
$(m_5 = ok \wedge m_4 = ab, m_5 = ab \wedge m_4 = ab)$

Rank 2 At this stage, rank 2 analysis can start: combining the results of extend at rank 1, all extensions of the partial fault modes contained in TODO$_2$ are computed (see figure 4).

Examining the six pairs of partial fault modes assigning m_2 and m_3, the algorithm can determine that they are all diagnosable except $m_2 = ok \wedge m_3 = ab$ and $m_2 = ab \wedge m_3 = ok$. The relative extensions do not mention any other mode variables, therefore the algorithm concludes that refinements of $(m_2 = ok \wedge m_3 = ab)$ are not discriminable from refinements of $(m_2 = ab \wedge m_3 = ok)$.

Examining the 4 pairs of partial fault modes assigning m_5 and m_1 or m_4, the algorithm can determine that $m_5 = ok \wedge m_1 = ab$, $m_5 = ab \wedge m_1 = ab$, $m_5 = ok \wedge m_4 = ok$ and $m_5 = ab \wedge m_4 = ok$ are diagnosable. On the other hand, the pairs $(m_5 = ok \wedge m_1 = ok, m_5 = ab \wedge m_1 = ok)$ and $(m_5 = ok \wedge m_4 = ab, m_5 = ab \wedge m_4 = ab)$ are not discriminable, their extensions mention respectively m_4 and m_1. The algorithm puts in the TODO$_3$ set the following combinations:

$(m_5{=}ok \wedge m_1{=}ok \wedge m_4{=}ab\ ,\ m_5{=}ab \wedge m_1{=}ok \wedge m_4{=}ok)$
$(m_5{=}ok \wedge m_1{=}ok \wedge m_4{=}ab\ ,\ m_5{=}ab \wedge m_1{=}ok \wedge m_4{=}ab)$
$(m_5{=}ok \wedge m_1{=}ab \wedge m_4{=}ab\ ,\ m_5{=}ab \wedge m_1{=}ab \wedge m_4{=}ab)$

In fact, pairs with different values for m_1 or m_4 are discarded due to property 1 (m_1 and m_4 being diagnosable, these combinations are in DISC$_3$).

Rank 3 By checking the observable variables for each pair (see figure 5), the algorithm finds that the 1st and the 3rd pairs are discriminable which makes the contained partial fault modes diagnosable, while the 2nd pair is not discriminable. Since the extensions do not constrain other fault modes, all the pair refinements are not discriminable as well. Therefore the algorithm stops at rank 3.

Results Two partial fault modes are not discriminable if and only if they refine the pairs $(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ok)$ or $(m_1 = ok \wedge m_4 = ab \wedge m_5 = ok, m_1 = ok \wedge m_4 = ab \wedge m_5 = ab)$. Partial fault modes that do not refine any of the 4 above are diagnosable. In terms of fault modes, 20 fault modes are not diagnosable (80 non discriminable pairs can be built), and 12 are diagnosable.

pfm	mode variables					observable variables				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$(m_2 = ok \wedge m_3 = ok)$	$*$	ok	ok	$*$	$*$	$*$	ok	ok	$*$	$*$
$(m_3 = ok \wedge m_3 = ab)$	$*$	ok	ab	$*$	$*$	$*$	ab	ab	$*$	$*$
$(m_2 = ab \wedge m_3 = ok)$	$*$	ab	ok	$*$	$*$	$*$	ab	ab	$*$	$*$
$(m_2 = ab \wedge m_3 = ab)$	$*$	ab	ab	$*$	$*$	$*$	ok	ab	$*$	$*$
$(m_5 = ok \wedge m_1 = ok)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
	ok	$*$	$*$	ok	ok	ok	$*$	$*$	ok	ok
$(m_5 = ab \wedge m_1 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ok	$*$	$*$	ab	ab	ok	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_1 = ab)$	ab	$*$	$*$	ok	ok	ab	$*$	$*$	ok	ok
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_1 = ab)$	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
	ab	$*$	$*$	ab	ab	ab	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_4 = ok)$	$*$	$*$	$*$	ok	ok	$*$	$*$	$*$	ok	ok
$(m_5 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_4 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
$(m_5 = ab \wedge m_4 = ab)$	$*$	$*$	$*$	ab	ab	$*$	$*$	$*$	ab	ab

Figure 4: The admissible extensions of rank 2 partial fault modes contained in the TODO_2 set.

pfm	mode variables					observable variables				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$(m_5 = ok \wedge m_1 = ok \wedge m_4 = ok)$	ok	$*$	$*$	ok	ok	ok	$*$	$*$	ok	ok
$(m_5 = ab \wedge m_1 = ok \wedge m_4 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
$(m_5 = ok \wedge m_1 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
$(m_5 = ab \wedge m_1 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ab	ok	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_1 = ab \wedge m_4 = ab)$	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_1 = ab \wedge m_4 = ab)$	ab	$*$	$*$	ab	ab	ab	$*$	$*$	ab	ab

Figure 5: The admissible extensions of rank 3 partial fault modes contained in the TODO_3 set.

Related work

The decentralized diagnosability approach proposed in this paper is based on the diagnosis algorithm developed in (Ardissono *et al.* 2005; Console, Picardi, & Theseider Dupré 2007) and fits in the same application context and reuses some of its contents.

Distributed diagnosability analysis is also studied in (Pencole 2004; Schumann & Pencole 2007), but the approach focuses on discrete event systems modeled by communicating automata. Diagnosability is analyzed in the context of event-based diagnosis, which means that the fault signatures are composed of sequences of events. In the discrete event systems framework, other well-known methods for diagnosability analysis, in particular (Sampath *et al.* 1995), but also (Cimatti, Pecheur, & Cavada 2003), (Jiang *et al.* 2001), are devoted to centralized systems.

In (Struss & Dressler 2003), diagnosability is analyzed upon a logic based formalism and within a state-based diagnosis framework that is closer to ours, however it focuses on centralized systems. The diagnosability analysis proposed in (Travé-Massuyès, Escobet, & Olive 2006) also refers to a state-based diagnosis approach for centralized systems. It uses a structural analysis approach and it is oriented towards continuous analytical systems, focusing on the problem of selecting the necessary and sufficient sensors/monitors to guarantee diagnosability properties.

In the field of continuous analytical systems, diagnosability analysis is often brought back to detectability issues and the design of fault indicators (residual generators) (Basseville 2001) (Staroswiecki & Comtet-Varga 1999) and the isolability problem is not the main focus.

Diagnosis and diagnosability have also been addressed in the system test community (Simpson & Sheppard 1994). A main difference of our work is that we do not assume that faults have independent symptoms, i.e. that the symptoms (abnormal observations) for a multiple fault are the union of the symptoms for the individual faults. Our analysis, based on extension of partial fault modes, is precisely addressed at identifying those consequences of faults that are independent from the presence of other faults.

Finally, the definitions of faults, fault modes, signatures and discriminability inspired from works bridging different diagnosis approaches (Cordier *et al.* 2004; Cordier, Travé-Massuyès, & Pucel 2006; Struss & Dressler 2003) are likely to suit most diagnosis approaches.

Conclusions and future works

The principle of performing the analysis by ranks is interesting for two reasons. First, the designer might be interested in diagnosing only a certain rank of faults, for example only rank one or two. In this case, one can stop the analysis at the corresponding rank, without analyzing higher ranks. Di-

agnosability analysis is a computationally expensive operation, and the designer might want to focus the analysis only on critical or most probable situations. A second reason is that a result involving a partial fault mode impacts on several situations, since it compactly represents many system behavioral modes. The designer can look at rank 1 results in order to have a very rough system analysis, and, if this is not sufficient, proceed to higher rank results. This allows in most cases to reduce the computational effort generally needed for a complete diagnosability analysis.

The diagnosability analysis presented in this paper does not provide information on how external inputs (e.g. inputs from users or other services outside the scope of diagnosability) affect the results. Future work will address this issue, by extending the analysis to find possible combinations of external inputs under which non discriminable fault modes become discriminable. For example, some fault modes might become discriminable assuming that the user does not cancel a requested action. Concerning the algorithm, future work will investigate the possibility of strengthening the properties in order to further reduce the search space.

Acknowledgements

This work was partially supported by the EU, grant IST-516933, project WS-DIAMOND (WS-Diamond 2005).

References

Ardissono, L.; Console, L.; Goy, A.; Petrone, G.; Picardi, C.; Segnan, M.; and Theseider Dupré, D. 2005. Enhancing Web Services with diagnostic capabilities. In *Proceedings of the 3rd IEEE European Conference on Web Services*.

Basseville, M. 2001. On fault detectability and isolability. *European Journal of Control* 7(8):625–637.

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. *Proceedings of IJCAI'03* 363–369.

Console, L.; Picardi, C.; and Theseider Dupré, D. 2007. A framework for decentralized qualitative model-based diagnosis. In *Proc. 20th Int. Joint Conference on Artificial Intelligence, IJCAI-07*.

Cordier, M.-O.; Dague, P.; Lévy, F.; Montmain, J.; Staroswiecki, M.; and Travé-Massuyès, L. 2004. Conflicts versus analytical redundancy relations : A comparative analysis of the model-based diagnostic approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man and Cybernetics - Part B.* 34(5):2163–2177.

Cordier, M.-O.; Travé-Massuyès, L.; and Pucel, X. 2006. Comparing diagnosability in continuous and discrete-event systems. In *Proceedings of the 17th International Workshop on Principles of Diagnosis, DX'06*, 55–60.

Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

Jiang, S.; Huang, Z.; Chandra, V.; and Kumar, R. 2001. A polynomial time algorithm for diagnosability of discrete event systems. *IEEE Transactions on Automatic Control* 46(8):1318–1321.

Pencolé, Y., and Cordier, M. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence Journal* 164(1-2):121–170.

Pencole, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *European Conference on Artificial Intel ligence ECAI?04*, 43–47.

Rozé, L., and Cordier, M.-O. 2002. Diagnosing discrete-event systems : extending the "diagnoser approach" to deal with telecommunication networks. *Journal on Discrete-Event Dynamic Systems : Theory and Applications (JDEDS)* 12(1):43–81.

Sampath, M.; Sengputa, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketsis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40:1555–1575.

Schumann, A., and Pencole, Y. 2007. Scalable diagnosability checking of event-driven systems. In *20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 575–580.

Simpson, W., and Sheppard, J. 1994. *System Test and Diagnosis*. Kluwer Academic Publishers.

Staroswiecki, M., and Comtet-Varga, G. 1999. Fault detectability and isolability in algebraic dynamic systems. *Proceedings of the European Control Conference*.

Struss, P., and Dressler, O. 2003. A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics. In *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*.

Travé-Massuyès, L.; Escobet, T.; and Olive, X. 2006. Diagnosability analysis based on component supported analytical redundancy relations. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans, Vol. 36, N6*.

WS-Diamond. 2005. Web-service diagnosability, monitoring and diagnosis. http://wsdiamond.di.unito.it.

Validation of a Multi-Agent Architecture for Planning and Execution

Maria D. R-Moreno[†], Guillaume Brat[‡], Nicola Muscettola[*] and David Rijsman[‡]
[†] Dpto. Automática, Universidad de Alcalá, Madrid, Spain.
mdolores@aut.uah.es
[‡] MCT, NASA Ames Research Center. CA 94035, USA.
brat@email.arc.nasa.gov
[*] Lockheed Martin. CA 94055, USA.
nicola.muscettola@lmco.com

Abstract

Traditional autonomous architectures are composed of technologically diverse functional layers operating at different levels of abstraction which makes the integration and validation difficult. In this paper we present the validation procedure followed in IDEA (Intelligent Distributed Execution Architecture).

IDEA is based on Multi-Agent system, where each agent relies on a domain model, a plan database, a plan runner, and a reactive planner. We have integrated a simulator agent into the architecture to validate the whole system behaviour. The simulator agent shares the physical agent model but extended to consider all the possible cases that can occur in the physical layer.

We also present how model checking techniques can be used to explore the space of input scenarios to validate the reactive planner with the simulator agent. The space of possible failures and reactions is explored and analyzed to define a set of scenarios that trigger realistic uses of the reactive planner. Our goal is to provide a good coverage of representative reactions to off-nominal behaviours of the environment being controlled.

Introduction

Autonomous control software uses models to reason about the system that it controls and the environment it is in. It accomplishes a set of goals during a period of time, and it is able to reason about failures with little or no human supervision. Given the initial state and external goals, it generates a set of synchronized low-level activities that, once executed, will achieve the goals. If any action is not executed as expected, the system is able to recover in order to achieve the pre-defined goals.

So far, the use of autonomous control software has been limited to controlling research rovers and some satellites such as EO-1. The biggest obstacles to its deployment in high-profile missions are its lack of real experience (despite the DS-1 experiment) and the apparent difficulty in validating it. Testing effective and robust control strategies requires dealing with any path of execution in any component and any module. In traditional systems, this already implies searching through a huge state space. Because of their flexibility and their reactivity to unpredictable events, autonomous systems generate even bigger state spaces; and, unpredictability makes mission managers nervous. Yet there is no denying that autonomous control software could bring greater flexibility, and most probably result in greater science returns than traditional control software. So what should we do? The answer is simple: we need to design architectures for autonomous control software with validation in mind.

In this paper we present how the IDEA Agent-based architecture allows a whole system integration and model verification. The domain model shared by the planners describes the operational mode of the system. We decided to validate this system by developing a simulator agent which is also controlled by a planner. The domain model of this planner consists of the domain model in the IDEA system and a set of constraints describing how the environment can react to the commands generated. This set of constraints includes success criteria for the commands (which models nominal operational scenarios for the underlying system) and failure modes (which models off-nominal scenarios). This simulator interacts with the planners in IDEA to simulate the reactions of the underlying system and the environment in which it will operate. Model checking techniques are used to explore the space of input scenarios to validate the reactive planner with the simulator agent.

The paper is structured as follows. The first Section provides an overview of the architecture for Planning and Execution. Then, we describe in detail the structure and behaviour of the Agent Simulator. Next, we revise the Model Checking Techniques integrated in the Agent Simulator. After, we show a simple example of the approach. Finally, we present the works related to our approach and the conclusions and future works are outlined.

IDEA Architecture

IDEA is a model-based autonomy architecture, being the agent the core of the architecture and the Reactive Planner the control engine of IDEA.

Figure 1 shows the different elements that compose the architecture. For a more detailed description of the architecture, the reader can refer to (Aschwanden *et al.* 2006), (Finzi *et al.* 2004), (Muscettola *et al.* 2002).

- The Communication Relay: transmits data and events to other agents or systems.
- The Agent Relay: is responsible for responding to internally and externally generated events within an IDEA agent. It relays outgoing messages to Communication Relays and incoming messages to Reactors.
- The Reactive Planner: is the responsible to respond to internally and externally-generated events, data and the passage of time. As the default Reactor, the Reactive Planner receives all incoming messages not explicitly modeled to be associated with another Reactor.
- The Agent Timing Service: provides a clock to synchronize IDEA agents.
- The Plan Service Layer: is the module that ensures complete consistent synchronization of the execution context between the Agent Relay and the internal state of the Reactive Planner. The (PSL) database stores and updates the past, present and future state of objects in the domain.
- The Goal Loader: is designed to import externally-defined goals (a temporally flexible plan) into the PSL.

At the heart of each IDEA agent is a declarative model describing the system that it can control, the activation of deliberative planning and the interactions with other agents.

IDEA models are declarative descriptions of legal operations of a system encoded in an XML based language *XIDDL* (XML IDEA Domain Definition Language). It allows us to describe interacting states and activities that can occur in complex systems in the same way as its predecessor DDL (Muscettola 1994). The main elements are:

- Class: is defined as a set of Timelines.
- Timeline (TL): is a logical structure used to represent states over time. A state is captured in a token. We can distinguish 3 modes of TLs:
  - Internal mode: describes the internal state of the controlled subsystem. State transitions are not communicated outside the agent and they are the result of compatibilities.
  - Goal mode: it represents a system which exerts control over the agent itself. State transitions can be the result of internal compatibilities or can be received from external systems. The agent will communicate return arguments and status parameters to the outside world (see parameters below).

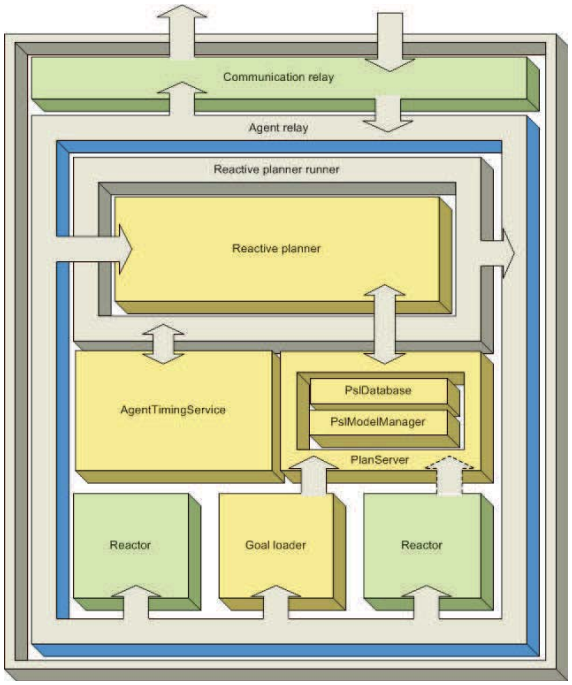


Figure 1: IDEA Architecture Overview.

  - Executable mode: describes the state that will be communicated to the outside world when the state transition is at the current time. This is also refered to as *dispatching* a command to the external system. The commands can be open or closed loop depending on the parameters defined in the model for the token. A closed loop command is a command for which the dispatching system expects a status back.
- Tokens: define the possible values that TLs can have. They can have zero or more parameters of the following types. Note that the the semantics of this type are applied for *executable* TLs. However, the semantics are inverted if these parameters are *goal* TLs.
  - `call_args`: this type of parameters are passed to the external system. They are expected to be determined before the command is sent to the external system.
  - `internal_modes`: this parameter is used for internal reasoning and not to communicate out.
  - `return_args` and `return_status`: these values are returned by the external subsystem. That is, the dispatching agent is expecting feedback from the external system. We can model as many `return_args` as we need but we can only define one `return_status` per token. The `return_status` parameter requires the defini-

tion of a boolean variable that is used internally by the Agent. If the value is *True* it means that a status from the exterior is expected to be received indicating the command is closed loop. If the value is *False* it means that we do not expect a status, also referred to as open loop. The agent will enforce the values of these boolean flags during execution, that is if a status is actually received the flag is forced to be *True*, if the external system does not return a status at the time the dispatching agent expects it the flag is forced to be *False*.

- Constraints: temporal and parameter restrictions between tokens.

The tokens are executed only after they have appeared in a plan maintained in a central database. This can happen either because a controlling agent has communicated new goals or because some internal planner (reactive or deliberative) has generated appropriate subgoals. To be considered for execution, a token must allocated on an appropriate TL.

The Simulator Agent

In order to verify and detect inconsistencies in the model and verify the whole architecture, we use another IDEA agent with a mirror model of the original model. The goal is to guarantee that the original model can handle most paths of execution of the external systems. Because generating all possible paths it may be infinite in complex models.

For that, we need to simulate different scenarios to cover the possible situations that can occur. Having in mind that the dispatching system - the Agent - sends out commands and can require return values or status back from the external system, and that the Agent can receive commands to execute a goal, we can use this mechanism to simulate any kind of behaviour including the functional layer systems.

For building an agent simulator we need to define the model that allows us to define its behaviour. The simulator model will be based on the original model such that the assumptions of the external systems captured in the compatibilities are also in the simulator model.

Figure 2 shows the different elements in the model that compose the agent and the simulator agent. The box on the left represents the domain model that drives the agent behavior. It contains the Physical Model, that is the objects, TLs, tokens and compatibilities that define the system we want to model, and the non-Physical Model can contain the TLs, objects, tokens and compatibilities not directly related to external systems such as activating deliberative planning or load goals. The box on the right represents the agent simulator declaration. Both the agent and the simulator agent share the Physical model but the mode of the TLs is inverted. That is, a *goal mode* TL will be converted into an *executable mode* TL and the *executable mode* TL into a *goal mode* TL; *internal mode* TLs will remain the same as they are intrinsic to each Agent. The TLs in the simulator side are "mirrors" of the TLs in the agent side.

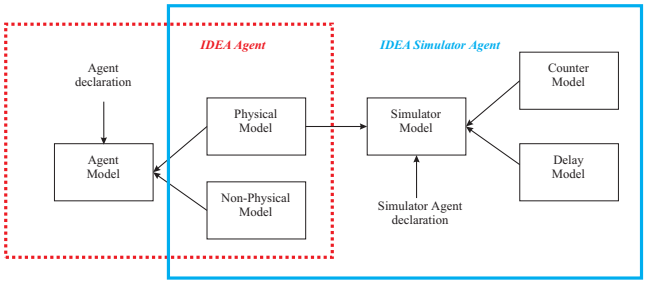


Figure 2: Agent and Agent simulator model structure.

The Delay Model refers to the gaps that will appear due to delays in the communication channel in each TL.

The Counter Model is the heritage from the simulation files that are used. Before the implementation of the Simulator Agent, the tests were created by hand using a non very intuitive syntax based on numbers and name tokens. For example, if there is a token called GPS with three arguments that refer to the x, y, and z position of the rover; and we want that after 2 time units the values of those parameters (i.e 3.14, 23.14, 13.14) are sent to the agent, the syntax used is as follows:

```
GPS    4   3   1
3.14   2   0
23.14  2   1
13.14  2   2
2      OK
```

The first row represents the token name, the ocurrence of the token in the plan (that is, the forth time the token appear in the database, will have the values set in the example), the number of parameters and an integer representing if we have or not a return status (0 means no, 1 means yes). Note that the index of the parameters starts by zero (last column in each parameter). The second column represents the time units when those values are sent. Finally, the last row represents when the return status value is sent and its value. Then, introducing errors in the creation of these tests is quite easy because the consistency of the values respect to the model are not checked.

So the Counter Model in Figure 2 will count the number of times that each token appears in each TL (second number in the first row of the simulation file) so we can have control over the commands we receive. The behaviour of the agent simulator can be split in two categories.

- Responding to dispatched commands: sending back the values of parameters and status of each dispatched command in case `return_args` and `return_status` have been defined in the model for a token on a *executable* TL. For example, in the real rover model we are working on, there is a token called *Drive* that belongs to the *executable* TL *DrivingQueryTL*. This token has 1 `call_args`, *DriveTo*, 1 `return_args` that refers to the location it drove to, *DroveTo*, and 1 `return_status` parameter indicating if the external system considered the command to be executed successfully. Its possible values are *Success* or *Fail*.

  The agent simulator has to decide how to respond each time it receives a *Drive* goal. For example it receives a *Drive* goal with call parameter value *ROCK1* for *DriveTo*. The simulator agent now has to decide when to send back a status, what value for the status and what value for *DroveTo*. Based upon the model the simulator should know what the allowed values are for these parameters given the context of the *Drive*.

  The time and the values will be selected using model checking techniques as explained in the next section. To do this we need to keep track of how many times we receive a command. We do this by enhancing the Physical Model with the Counter Model.

- Dispatching commands to the agent: the simulator is responsible for deciding the timing of the commands, what commands and what the values are for the call parameters. For the receiving agent these commands are goals and once received it has to find a plan consistent with the new goals.

Figure 3 shows the integration in the Simulator Agent of the counter and delay models for an *executable* TL in an agent. The *Tk1* token in the Agent Side sends its start time to the mirror TL in the Simulator Side which receives it with a delay. The same process occurs when the token finishes. Then, the *meets* compatibility triggers the start of the *Tk2* token. Here the situation varies because of the status value. The Counter TL in the Simulator side (defined as an *Internal* TL) checks the occurrence of the token in the TL and sends the value(s) back to the Agent side. Let us say that in the Agent side there is a compatibility to meet the token *Tk3* if the status is for example Fail. In the Simulator Side no decisions should be made until the next start message is received (token *Tk3*). Then a *Buffer* token is added in both counter and mirror TLs to fill the gap due to the delay in receiving the next start time token.

Then, we have built the appropiate environment to generate the tests compatible with the model. The next step is to decide: what do we want to test, and what values do we want to set. Next section describe how this could be done.

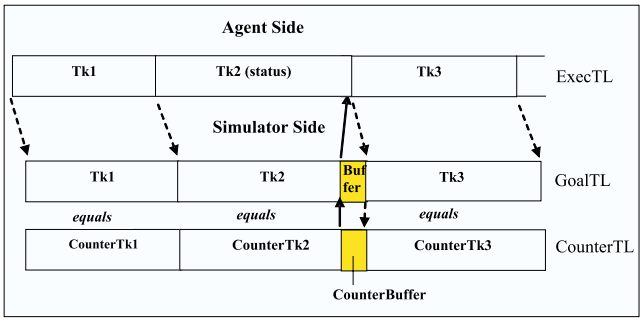


Figure 3: Counter and Delay Models design.

Model Checking Techniques

This section describes how we generate scenarios for the simulator agent. On one hand, simulating nominal behaviour is a simple matter of going through combinations of commands. On the other hand, off-nominal is trickier since it involves guessing what environmental conditions or mishaps in the underlying physical system can trigger reactive planning. Our approach consists of modelling the space of off-nominal actions and using model checking to generate a representative set of off-nominal scenarios.

Model-based test generation

Model checking is a method to verify finite state systems formally. This is achieved by verifying if the model (derived from the design phase or by abstraction of the code) satisfies a logical property (derived from the requirements). Properties are often expressed as temporal logic formulas, but simple assertions can also be checked.

The model is usually expressed as a directed graph consisting of nodes (or vertices) and edges. A set of atomic propositions is associated with each node. The nodes represents states of a program, the edges represent possible executions which alters the state, while the atomic propositions represent the basic properties that hold at a point of execution. The problem can be expressed mathematically as: given a temporal logic formula p and a model M with initial state s, decide if:

$$< M, s > \models < p > .$$

In general, model checking explores systematically a state space while examining all possible choices at decision points. However, exhaustive search is often impossible to achieve. Fortunately, when properties are violated by the system (under exploration), the model checker can be content with simply returning a trace illustrating the violation (also called, a counter-example). This is very similar to the process of planning, during which a planner looks for and stops as soon as it finds a plan satisfying the planning constraints.

The process of creating counter-examples in model checking can be exploited to generate test cases guaranteeing a certain coverage for the system under test. The desired coverage criteria are gathered in a property (which can be as simple as a conjunction of assertions on the values of state variables) so that the violation of the property will cause the model checker to return a (counter-example) path that satisfies the coverage criteria. A test case can then be extracted from the trace by identifying the input values (and possibly their order and timing). This process may look redundant since the model checker is exploring the path that we want the test to exercise. However, meeting the coverage criteria (i.e., violating the property representing the criteria) does not necessarily imply a full path exploration. In this exercise, the model checker is really looking for a quick way to meet the criteria. This can actually be a problem in the sense that modern model checkers are getting too good at returning minimal counter-examples to illustrate violations. In our case, it means that we generate test cases that exercise very little of the system. This problem can be solved by allowing the model checker to start its exploration from arbitrary points in the state space; this gives us a sort of "bunny hopping" strategy to test generation.

SAL

SAL (Symbolic Analysis Laboratory) is a framework for combining different formal method tools to calculate properties of concurrent systems. The goal of SAL is to be scalable, automatic, and cost effective. SAL is really an intermediary language, developed conjointly by SRI, Stanford, Berkeley, and Verimag, which can be used to integrate various tools based on formal methods. In this work, we are using SAL 2 developed by SRI (de Moura *et al.* 2004).

For example, the contribution of SRI to the SAL framework deals with augmenting the PVS theorem proving framework with tools for abstraction, invariant generation, program analysis (such as slicing), theorem proving, and model checking. In particular, they use PVS to reduce a verification problem into a finite form and then model checking to calculate verification properties. SRI also integrates a bounded model checker for timed automata into the SAL framework; it gives them the ability to deal with models, or systems, constrained with time information. The current version of SAL also includes an infinite-bounded model checker which provides bounded model checking for systems defined over infinite data types, such as integers and reals. SAL can deal with LTL as well as CTL properties. SAL describes transition systems in terms of initialization and transition commands. These can be given by variable-wise definitions in the style of SMV or as guarded commands in the style of Murphi.

The SATELLITE Domain Example

In order to validate the ideas mentioned above, we have started by modeling in XIDDL a simple domain, the SATELLITE domain where our ideas have been tested. This domain belongs to the set of domains used in the 3rd International Planning Competition (IPC'02). Given a collection of satellites that contain different instruments in specific modes, the goal is to take pictures in some directions using the available instruments in the satellite. Let us resume the actions that belong to this domain:

- `Turn_to`(?s - satellite ?d_new - direction ?d_prev - direction): after the action is performed, the satellite points to a new direction.
- `Switch_on`(?i - instrument ?s - satellite): at the end of the action, the power is available in a specific intrument inside the satellite.
- `Switch_off`(?i - instrument ?s - satellite): is the opposite of Switch_on, that is, after the action is executed, the power in the instrument is off.
- `Calibrate`(?s - satellite ?i - instrument ?d - direction): once the satellite points to the target direction and the instrument is switched on, then the instrument is calibrated and ready to take an image.
- `Take_image`(?s - satellite ?d - direction ?i - instrument ?m - mode): if the instrument is calibrated and switched on, then the image is taken.

Figure 4 shows the Timelines (TL) considered for the SATELLITE domain. Basically, we have considered 4 TLs, each one representing the evolution in time of a physical component. In this case, we have abstracted the satellite, the power, the instrument, and the mode. These four TLs are considered as *executable* TLs since the tokens of the TLs can send messages to the outside world and may receive return and status values (see section to an explanation of the type of parameters tokens can have).

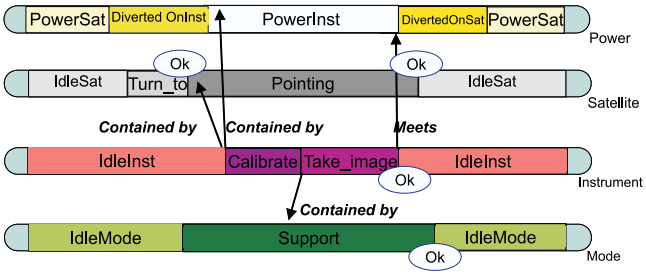


Figure 4: The SATELLITE domain using the Timeline representation.

For the purpose of the example, we are going to consider that only the `Turn_to`, `Pointing`, `Take_image` and `Support` tokens receive a status from the external world, which in our simulating environment, that value is generated by the simulator agent.

Since there cannot be gaps between tokens, we have added extra `Idle` tokens to all of the TLs. In the case of the Power TL, the `PowerSat` represents the Idle token and instead of the `Switch_on` and `Switch_off` operators, we have modelled the power status using the `DivertedOn (Sat/Inst)` and `PowerInst` tokens.

Tokens in the same TL are connected by the *meet* compatibility. The relationship with other TLs is defined using any other temporal relation. For example, the `Calibrate` token has to be contained by the `PowerInst` and `Pointing` tokens.

We now describe the model we used for the simulator input generation. We want to generate scenarios that can exercise the off-nominal scenarios. Therefore, we concentrate on the off-nominal behaviour of the underlying system and the environment. Yet let us start by describing model variables related to nominal activities. Each activity (or command) can generate a:

- status, such as **OK** or **Failed**,
- occurrence index, which indicate the occurrence of similar activities

Now, the off-nominal scenarios can come from the following places:

- message delays: each command and status reply go over the network, and therefore, may be subject to delays. Arbitrarily, for the sake of this example, we decided that delays can range from 1 second to 5 seconds.
- latency: the beginning of an activity might not correspond to the frequency interval at which the agent is controlling the system. The latency range is given by the frequency of the control.
- send and receive times: times at which message are emmitted and received by the agent and the simulator. It is closely related to the message delay.

The transitions in our model are given by the nominal behavior, which is given by the compatibilities in the Model of the agent. For example, the activity **TurnTo** is met by the activity **Pointing** on the **Satellite** TL whereas the activity **Calibrate** makes sense only for the **Instrument** TL and is met by a **TakeImage** activity. Note that this model also includes timing information for each activity. Now, the off-nominal scenarios are described by a state machine, which takes into account timing information such as message delays and activity durations. Different time values lead to different **status** state. For example, for the activity **TurnTo** on the **Satellite** timeline, a duration of 5 seconds leads to an **OK** state (corresponding to the nominal behaviour) while a duration of 7 seconds leads to a **Failed** state (corresponding to an off-nominal behaviour which will trigger the reactive planner).

Now, this model really needs to be expressed using the SAL language. Then the *sal-atg* tool can be used to generate test inputs (Hamon *et al.* 2004) (Hamon *et al.* 2005). This is achieved by adding trap (boolean) variables in state of interest. These variables are initially set to *FALSE* and are set to *TRUE* when their corresponding state is explored by the model checker. Now, the states containing trap variables are chosen in such a way that we hit all boundary conditions in the timing variables, giving us a coverage of all the scenarios in which timing values cause the agent to miss its deadlines. This allows us to generate scenarios such as "At occurrence zero of the **TurnTo** activity a duration of 7 seconds will cause the simulator to return a **Failed** status".

Although we have presented a simple example where there is just one agent, the IDEA architecture allows us to create as many agents as we need and then, simulator agents. In the XIDDL domain description we also have to provide (see the concepts introduced in section 2), the agent topology and configuration description, consisting of: the *latency*, a declaration of all of the known initial subsystems associated to that agent and the communications *channels* between the agents and the subsystems in the domain. Since we want to simulate the subsystems behaviour, the target channel is modified by the name of the simulator agent in charge of its behaviour. In the same way, in the simulator agent description, we write as the target channel of that subsystem the name of the normal agent. Like that, the communication between agents and simulator agents is guarantied.

Related Work

The Remote Agent (RA) (Muscettola *et al.* 1998) was based on a three-layer architecture - Functional, Execution and Planning/Scheduling - each one using different technologies which complicated the integration and testing of the whole system. Another complication of the layered architectures is the lack of access from the Planner to the Functional Level. Most of the traditional autonomous and robot architectures follow this approach, differing in the degree of dominance of some layers over the others (Estlin *et al.* 1992), (Borrelly *et al.* 1998), (Estlin *et al.* 2000).

More recent approaches, as CLARAty (Volpe *et al.* 2001), try to overcome some of these drawbacks using a two-layer architecture: the Functional and the Deliberative layers. The Functional layer follows an object oriented design for the hardware components. The Deliberative layer integrates planning and execution through a tightly-coupled Database that allows synchronizing the same information using two different representations: the one from CASPER (planning) and the one from TD (execution). The simulation of the system consists of method calls that emulate some behaviour. In IDEA, the planning, execution control and simulation share the same modelling framework what makes the integration and validation an easier task. In the model, it is possible to abstract what to simulate and error injection can be easily incorporated.

Another two-layer architecture is the model-based approach pursued by (Williams *et al.* 2004). This

framework consists of a Reactive Model-Based Programming Language (RMPL) and an executive (Titan). A model-based program specifies two inputs, a control program and a system model. The language abstracts from the functional level so the programmer can reason in terms of state variables not directly corresponding to observable or controllable states, which is different from IDEA where the model is directly integrated with the functional level. From the simulation point of view, RMPL cannot provide time related utilities such as timeout definitions or warping because of the state variables representation. The simulation is done synchronously and backward from observations. Our architecture instead, provides any time related utilities and richer scenarios than just the ones from observations.

Conclusions and Future Work

In this paper we have presented an architecture for planning and execution based on Multi-Agent systems. An agent can communicate with multiple agents both controlling and controlled. Although two agents could mutually control each other. The allowed communications are governed by a Model that describes which procedures can be exchanged with which agents.

Within this framework we have integrated a Simulator Agent that can simulate some behaviour. The goal is to guarantee that the model can handle all paths of execution. For that we have simulated different scenarios to cover all the possible situations that can occur. Model checking techniques are used to explore the space of input scenarios to validate the reactive planner with the simulator agent.

The approach has been first tested in a simple satellite model from the IPC'02 competition. Then, in the real rover model Gromit (Finzi *et al.* 2004). Thanks to the simulator agent we have automated the simulation process (before the files were written by hand) and created tests that the testing group haven't thought about them.

For the future we also want to use co-evolution techniques to optimize the scenarios generated under some stress parameters, i.e. reactive time or time between deliberative and reactive planning. And we also want to study if probabilistic planning can be integrated in this architecture.

Acknowledgements

R-Moreno's work at NASA Ames has been supported by the Spanish Ministry of Education and Sciences. Nicola Muscettola's work was conducted while he was at NASA Ames. This work has been partially funded by the Castilla-La Mancha project PAI07-0054-4397 and the CICYT project ESP2005-07290-C02-02.

References

Pascal Aschwanden, Vijay Baskaran, Sara Bernardini, Chuck Fry, M.D. R-Moreno, Nicola Muscettola, Chris Plaunt, David Rijsman, and Paul Tompkins. Model-Unified Planning and Execution for Distributed Autonomous System Control. In *AAAI 2006 Fall Symposia. Washington DC (USA), October*, 2006.

J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The ORCAD Architecture. *International Journal of Robotics Research.*, 17(4), 1998.

L. de Moura *et al.* SAL 2. In *Procs. of Computer-Aided Verification, CAV 2004*, 2004.

T. Estlin, G. Rabideau, D. Mutz, and S. Chien. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. *The Journal of Real-Time Systems,*, 4(1):37–53, 1992.

T. Estlin, G. Rabideau, D. Mutz, and S. Chien. Using Continuous Planning Techniques to Coordinate Multiple Rovers. *Elec. Trans. on AI*, 4:45–57, 2000.

A. Finzi, F. Ingrand, and N. Muscettola. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In *Procs. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan*, 2004.

G. Hamon, L. de Moura, and J. Rushby. Generating Efficient Test Sets with a Model Checker. In *Procs. of SEFM'04*, 2004.

G. Hamon, L. de Moura, and J. Rushby. Automated Test Generation with SAL. Technical Report CSL Technical Note, SRI, USA, 2005.

N. Muscettola, P. Nayak, B. Pell, , and B.C. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2):5–48, 1998.

N. Muscettola, G.A. Dorais, C. Fry, R. Levinson, and C. Plaunt. IDEA: Planning at the Core of Autonomous Reactive Agents. In *Procs. of the Workshop On-line Planning and Scheduling, AIPS 2002, Toulouse, France*, pages 49–55, 2002.

N. Muscettola. HSTS: Integrating Planning and Scheduling. In M. Zweben and M.S. Fox, editors, *Intelligent Scheduling*, pages 169–212. Morgan Kaufman, 1994.

R. Volpe, I.A.D. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty Architecture for Robotic Autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*, 2001.

B.C. Williams, M. Ingham, S. Chung, P. Elliott, and M. Hofbaur. Model-based programming of fault-aware systems. *AI Magazine*, 24(4):61–75, 2004.

Fault Diagnostic of Bearing Via Physics-Based Modeling Techniques

Amir Shirkhodaie

Intelligent Manufacturing Research Laboratory
Department of Mechanical & Manufacturing Engineering
Tennessee State University, 3500 John Merritt Blvd., Nashville, TN 37209
Email: ashirkhodaie@tnstate.edu

Abstract

Faults in rolling element ball bearings can be estimated using analytical and empirical methods or predicted grossly using artificial intelligence techniques such as expert systems, fuzzy logic, and neural network, etc. When the operational condition of a bearing is dynamic and there is a need to determine the state of stresses on the bearing concisely, then it is more appropriate to develop a physics-based model of the bearing based on finite element modeling (FEM) techniques and subjecting it to similar operational loading condition to discover its state of stresses. By applying such FEM techniques, one can in effect examine many possible mechanical characteristics of bearing under operational conditions, e.g., speed, temperature, and internal and external vibration, etc. Understanding such mechanical characteristics is crucial to accurate fault diagnosis of the bearings in practice. In this paper, we present our technical approach toward the development of a physics-based finite element model of rolling element bearings and provide some examples based on results of this research effort.

Introduction

The purpose fueling the technical research of a physics-based model of rolling element bearings using finite elements stems from the need to better understand the mechanical characteristics of the rolling element bearing. Rolling element bearings are in extensive use today since they develop less friction than plain bearings, which corresponds to lower power losses and lower operating temperatures, they are less sensitive to load fluctuations, require only small quantities of lubrication, occupy shorter axial lengths than plain bearings, and support combined radial and thrust loads. However, rolling element ball bearings often fail due to fatigue, which is an area of significant concern and has received extensive analysis by researchers.

The bearing life models have continued to improve through refinements in technology. Zaretsky [1996] utilized the finite element method to compare the Lundberg-Palmgren, Zaretsky-modified Weibull, and Ioannides-Harris rolling element life model theories by analyzing the critical stress-bearing life relationship. These theories approach the bearing fatigue life analysis based on the initiation or first evidence of fatigue spalling on either a bearing race or a rolling element only to predict life of a bearing design. Zhao [1998] combined a finite element model or virtual contact loading method (VCLM) with iterative boundary problem numerical methods to solve multi-body contact problems in roller bearings under quasi-static load conditions. Very few reports of the load distribution within a rolling element bearing exist, as Zhao reported. Bhargava [1985a] developed a single contact model and repeated contact model [1985b] of a cylindrical rolling element by translating a semi-elliptical pressure distribution over a finite element mesh representing a semi-infinite body to determine displacements and normalized von Mises equivalent stress levels. Kral [1996] conducted an analysis of surface deformation and stress levels due to indentation and sliding contact in an elastic-plastic medium using a three-dimension finite element model. Nonetheless, improvement in modeling the actual stresses and deformation of the internal bearing components is needed to assess bearing load conditions that will aid in accurately identifying bearing faults. Machine fault diagnosis and predictive maintenance are very important industrial activities and rolling ball bearings are very common in rotating component applications. However, rolling element ball bearings often fail due to surface fatigue, which is an area of significant concern, and has received extensive empirical analysis.

In fact, twelve primary causes of bearing failures have been identified to include: excessive loading, overheating, true brinelling, false brinelling, normal fatigue failure, reverse loading, contamination, lubrication failure, corrosion, misalignment, loose fit, and tight fit. Therefore, demand has increased for improved industrial machine designs and maintenance procedures that reduce operating costs and raise machine reliability coupled with enhanced machine performance that saves energy and raises operational safety. Development of an analytical bearing diagnostic system using a finite element model is viable since bearing failure modes are directly associated with stress and material deformation levels. The operational bearing acceleration and temperature signals are related to the internal bearing response to the applied loads. The bearing stress and deformation levels, in the finite element model, are correlated with the response relationships to actual bearing degradation and failure modes for early fault detection. Nonetheless, bearing failure detection and prevention is complex and multi-factor dependent. Therefore, these response relationships could serve as a “look-up” table for quicker bearing analysis.

The goal of this research entails applying finite element model

(FEM) methods to develop an accurate parametric model of a precision deep-groove ball bearing. This goal necessitates conducting a full factorial design of experiments (DOE), over the range of various load conditions that a bearing undergoes, to create a physics-based model of deep-groove ball bearings for the purpose of condition-based monitoring under various operational conditions. Accordingly, the research objectives include: creating a parametric FEM of physical deep-groove ball bearings and conducting a thorough FEM analysis of the bearing model under static and dynamic loads; developing a physics-based model of bearings using results obtained from the finite element analysis of bearing models; building an experimental bearing test bed to collect actual failure mode data for correlation with the physics-based finite element model; assessing the physics-based model in the diagnosis of bearing failure modes using quantitative and qualitative evaluation criterion; correlating the bearing failure mode data with the experimental bearing test bed and the physics-based finite element model to develop a performance "look-up" table.

Bearing Geometry

A bearing element structure consists of rolling elements, inner and outer rings, and housing. The finite element model test bearing was created using a standard pillow block mounted deep-groove Boston XL ball bearing with a 1-7/16" bore as the bearing model prototype. This bearing was selected since it possessed a moderate load rating, which is a common industrial application requirement, and was commercially available in quantity.

Bearing Raceways

The bearing has ten symmetrically spaced rolling elements that are 11/32" in diameter each as shown in Figure 1. It is presumed from the manufacturer's drawings that the bearing is designed according to industry-accepted design criteria for the initial finite element model. The radius of curvature of the inner and outer raceway should have 51.5% to 53% osculation with the rolling element surface and a diametral clearance, P_d, of 0.00045". Diametral clearance is the space between the rolling ball element and either raceway. These assumptions are critical to the way the bearing distributes the applied load and affects stress, deflection, and fatigue life. The diametral clearance is measured from the diameter and defined by the equation $P_d = d_o - d_i - 2D$, where d_i is the inner raceway diameter, d_o is the outer raceway diameter, and D is the rolling element ball diameter. The bearing pitch diameter, d_m, is approximately equal to the mean of the bore diameter, D_I=1-7/16", and the outside diameter, D_o = 1/2(D_I + D_o). More precisely, the bearing pitch diameter is the mean of the inner and outer raceway contact diameters. Therefore, using the approximation for d_m=2.125", the inner diameter is d_i= 1.781025" and the outer diameter is d_o= 2.468975". As shown in figure 3, the clearance between the raceway surface and the ball bearing is 1/4P_d = 0.0001125". The ability of a ball bearing to carry a load depends in large measure on the osculation of the rolling elements and raceways. Osculation, ϕ, is the ratio of curvature of the rolling element to that of the raceway in a direction transverse to the direction of rolling, where osculation is

$$\phi = \frac{D}{2r}$$

Using 52% osculation, the bearing raceway radius is calculated as r = 0.17875". Using the results from these calculations, the ball bearing geometry was created as shown in Figure 2.

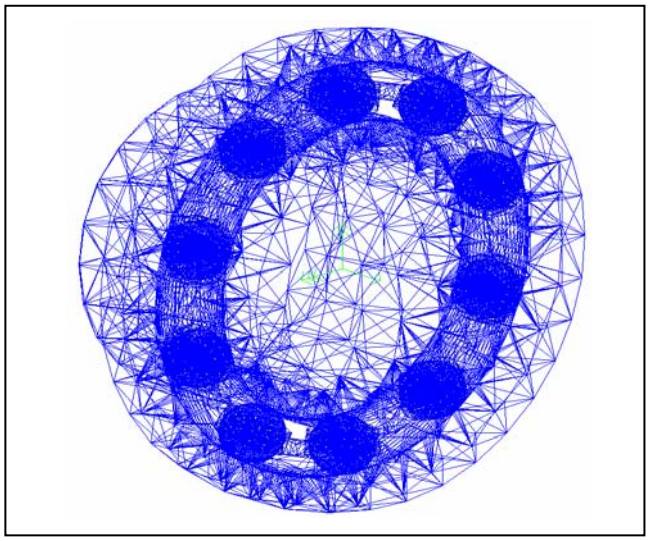


Figure1. Bearing Finite Element Model

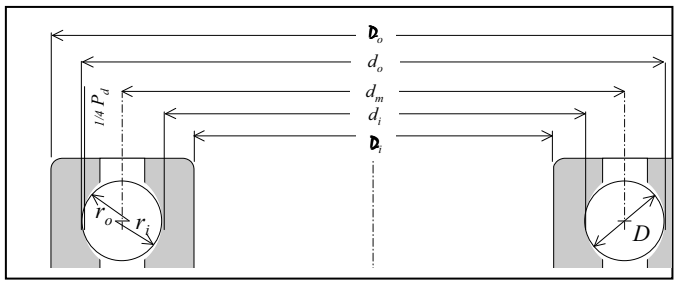


Figure 2. Ball Bearing Design Geometry

The MSC/ARIES finite element modeling software package incorporates a mathematical integration module that allows the user to specify "precise" solid geometry instead of using facetted geometry according to some tolerance.

This "precision" mode of geometry enhances the model accuracy, which significantly improves the finite element mesh created along the solid model curves by the manual or automatic mesh generator. The solid geometry of the bearing was generated using the curves shown above starting from the inner race and working outward to the outer race as shown in Figure 3.

Load Conditions And Restraints

Once the bearing model was completed, static radial and axial load conditions were applied to the bearing outer race. The shaded arrows shown in figure 5 depict the applied forces as they act on the surface of the outer race. The load conditions were selected within the acceptable load range of the design load bearing capacity. The radial load shown is a positive 0 to 500 lb_f force applied in along the vertical y-axis and the axial load is a 0 to 250 lb_f force applied along the x-axis in the negative direction. The inner race is restrained from translating in the x-axis, restrained from rotating about the y-axis and the z-axis, and free to rotate about the x-axis to simulate that the bearing inner race is mounted to a shaft. The load conditions and restraints must be specified prior to creating the finite element mesh, which will be covered in the next section.

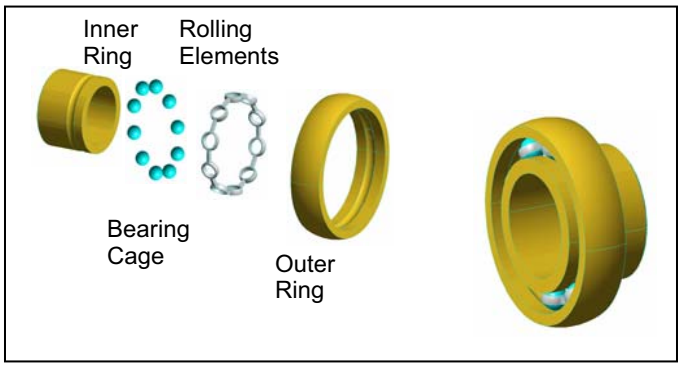


Figure 3. Boston XL Bearing Components

Assumptions

The solid geometry is considered perfectly symmetric and constructed of isotropic material free from internal defect. The bearing is installed with sufficient pre-load conditions to maintain constant contact between rolling elements and raceways. The lubrication film between the rolling elements and the raceway is three times greater than the material surface roughness to provide a pre-determined elasto-hydrodynamic (EHD) film layer thickness. The cage (or retainer) keeps the rolling elements symmetrically spaced about the rotational axis and the bearing raceways. The inner ring is considered mounted on a constrained shaft rotating at 1800 RPM and the loads are applied through the outer ring. The load conditions are within the linear-elastic range of the materials. The test loads are applied radially to the outer ring bottom surface and axially to the exterior circumference face as shown in Figure 4.

Finite Element Mesh Generation

Once the solid geometry was completed, with the specified load and restraint conditions, the finite element mesh was generated using both the manual meshing technique and the automatic mesh generator. Before meshing the solids with the automatic mesh generator, surface meshes were laid down on the critical surfaces to ensure that the finite element node points were aligned. This would become a critical operation later in the modeling process. The surface meshes were created along the inner raceway and the outer raceway with 40 node points along the peripheral diameter and 10 node points along the raceway radius using the quadrilateral mesh design. The ball bearings were created as spheres and each hemi-

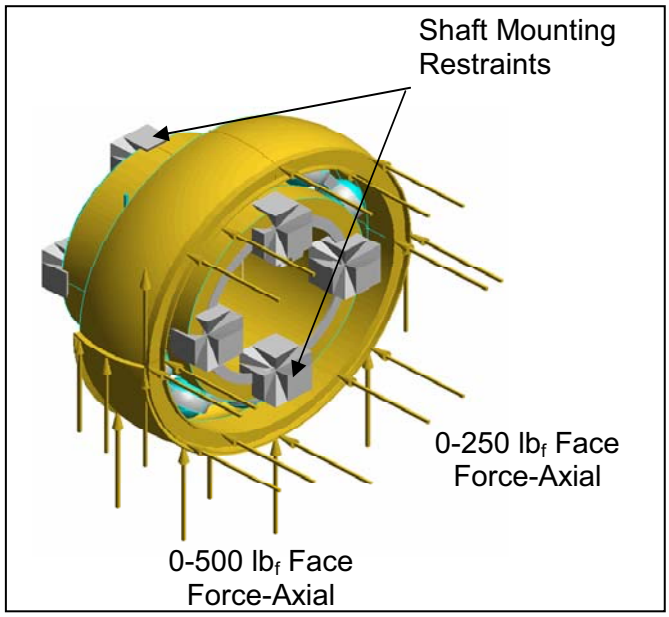


Figure 4. Boston XL Bearing with Load Conditions and Restraints

sphere was surface meshed with a two-point loft surface mesh containing 24 edge nodes and 6 elements. Upon completion the surface meshes, the automatic mesh generator was used to complete the solid geometry finite element mesh using a 0.25" default tetrahedral element with 30-degree automatic curved-based scaling. The mesh generator automatically reduces the

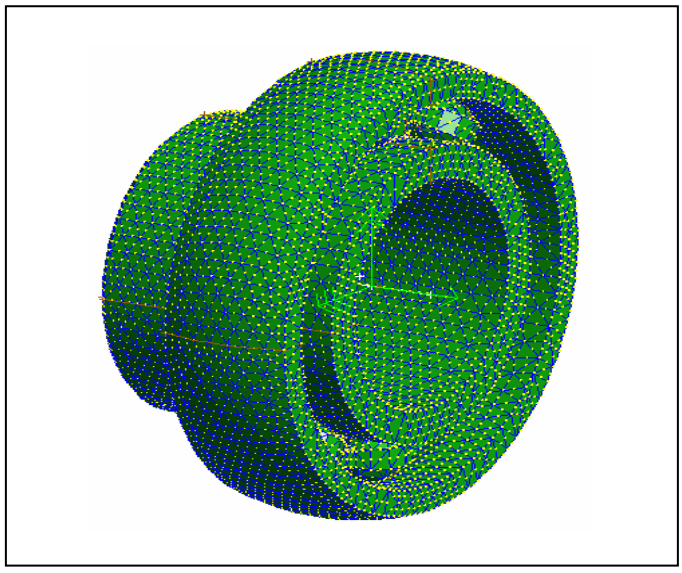


Figure 5. Deep-groove Ball Bearing Finite Element Model

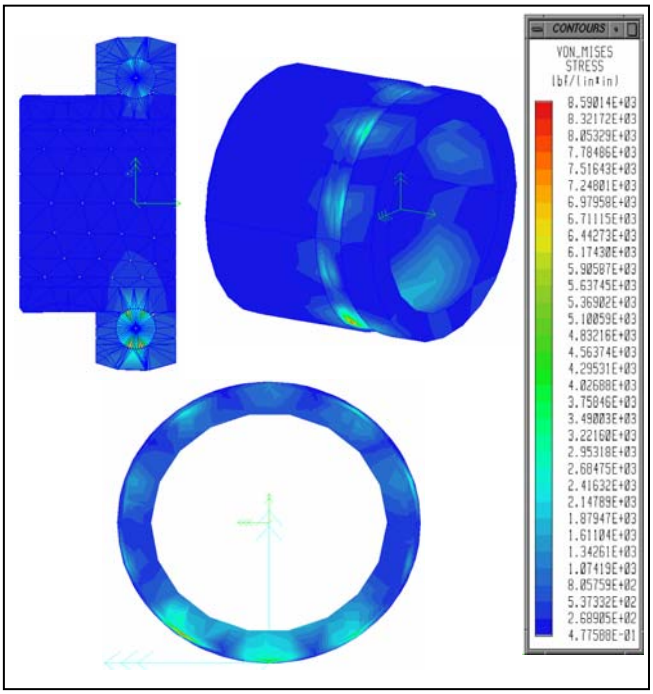


Figure 6. Finite Element Results of Stressed Bearing

element length to prevent elements from crossing solid geometry boundaries. The nodes along the ball bearing-raceway surfaces were merged to allow the force flow between the various bearing components. The completed finite element model contained 9,120 elements with 4,022 nodes. The load conditions and restraints were converted to nodal loads and nodal restraints at the finite element nodes where the load forces and restraints acted in the model. The assembled finite element model is shown in Figure 5.

Finite Element Results

The finite element model was processed using MSC/NASTRAN. Special attention was devoted to fine mesh size generation at the critical stress areas around the bearing. The analysis results of the bearing finite element model are shown in Figures 6, 7, and 8. After analyzing the previous finite element model, a revised finite element mesh was generated using both the manual meshing technique and the automatic mesh generator discussed previously. Whereas, the ball bearings meshes were created using a two-point loft surface mesh containing 24 edge nodes and 12 elements. This improved the accuracy of the ball bearing geometry without adversely increasing the number of elements and nodes in the model. The automatic mesh generator was again used to complete the solid geometry finite element mesh using a 0.25" default tetrahedral element with 30-degree automatic curved-based scaling technique.

The revised finite element model contained 16,243 elements with 10,373 nodes. Also, the radial and axial load conditions were applied individually and later combined in the finite element results analysis to allow scaling of the load conditions and minimize computer analysis time.The rolling element ball bearing deformation associated with the applied radial loads from 0 to 500 lb_f (in 100 lb_f increments) and the applied axial loads 0 to 250 lb_f (in 50 lb_fincrements) was determined for future correlation. It is assumed that the material is isotropic and the bearing-raceway geometry is perfectly symmetric. The load cases selected are within the rated load capacity of the bearing at 1800 RPM. The graph in Figure 10 details the corresponding deformation in inches for the given radial and axial load conditions.

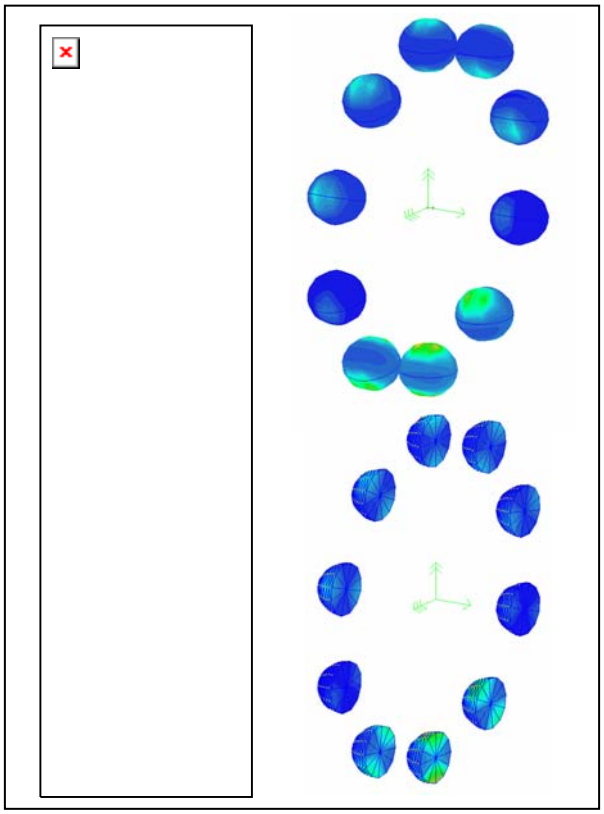


Figure 7. Bearing Finite Element Results

Bearing Element Contact Geometry

Dubeck and Shirkhodaie [1999] previously discussed the bearing geometry in detail, however, the contact geometry deserves additional attention. Hamrock [1983a, 1999] developed a simplified means to determine the loads (within +/- 3%) carried by the bearing and transmitted through the rolling elements from one race to the other. The magnitude of the load carried by an individual rolling element depends on the internal geometry of the bearing and the location of the rolling element at any instant. When the elastic rolling element solids are brought into contact with the races under an applied load, an elliptical contact area develops. The finite simplified solution developed by Hamrock [1999]. Figure 11

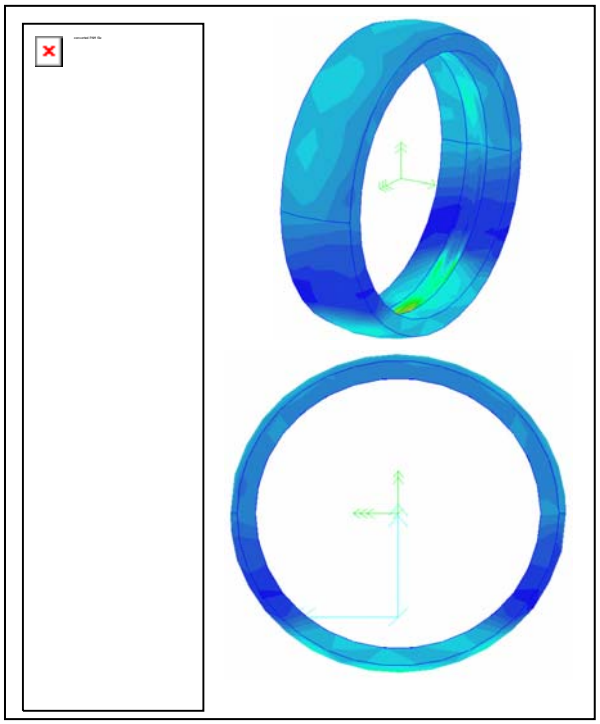


Figure 8. Outer Race Finite Element Results

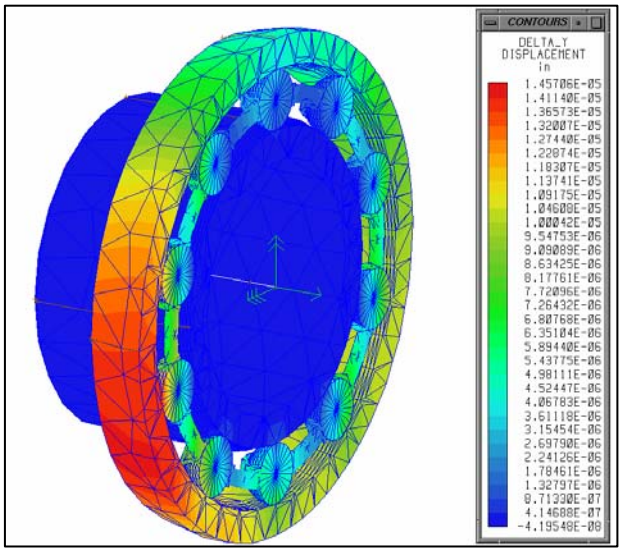


Figure 9. Strain Profile of the Stressed Bearing

depicts elliptical contact area associated with a loaded ball bearing. The analytically computed ball element displacement and the elasto-hydrodynamic lubrication film thickness at the inner and outer race contacts are shown in the Figure 12. This information was taken into account at the time of development of the finite element model.

Analytical Dynamic Finite Element Results

Gupta [1977, 1985] developed equations of motion for the natural frequencies of ball bearing rolling elements and pointed out that the "elastic contact frequency" can play a dynamic role at the outer race that may contribute to the contact load when the inner race serves as a contact spring that provides a static force. As shown in Figure 13, the elastic contact frequency varies non-linearly at almost 1/6 power with the applied load. Combining the rotational displacement data measured at the outer race with the elastic contact frequency, the computed displacement signal appears periodical as shown in the Figure 14.

A discrete Fourier transform was applied to this signal to calculate the displacement magnitudes and frequencies at the outer race housing for increasing radial loads from 0 lb_f to 600 lb_f at 1800 RPM as shown in the Figure 17. The frequency response function of the model ball bearing under increasing

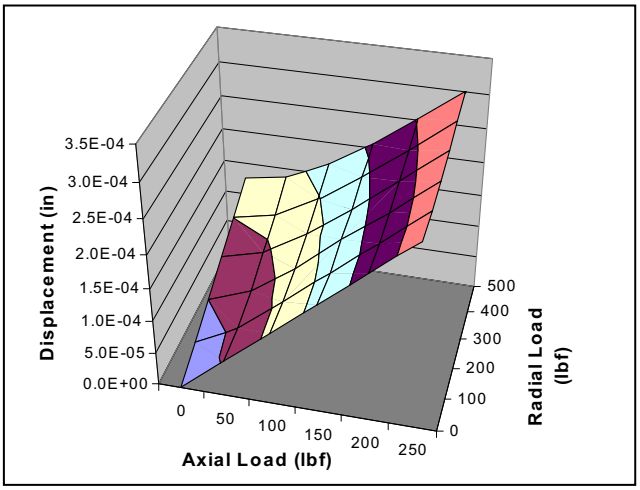


Figure 10. Axial & Radial Load Conditions with Deformation Results

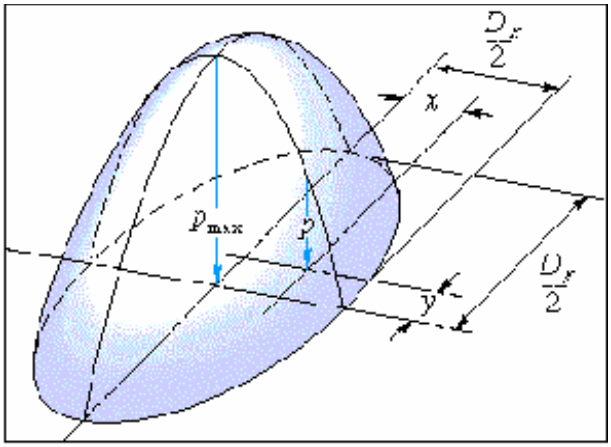


Figure 11. Rolling Element Contact Area

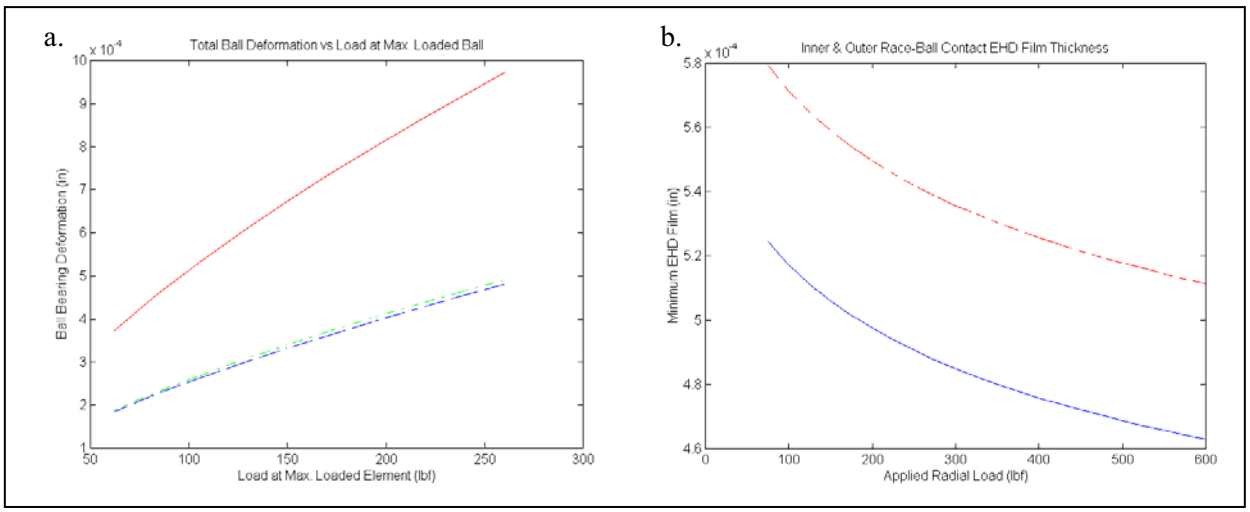


Figure 12. (a) Rolling Element Deformation, (b) EHD Lubrication Film Thickness

applied radial loads and running at 1800 RPM is shown in the Figure 15. This technique serves us as a means to approximate actual operating load conditions of the rolling elements and identify internal bearing defects.

Bearing Experimental Testbed

We have designed and constructed a bearing experimental testbed to explore the static and dynamic responses of the test bearings shown in Figure 16. The bearing experimental testbed operates on a 10Hp AC drive motor and facilitates us to obtain experimental results on vibration, deflection, temperature, and acoustic characteristic information for each set of test bearings and any variety of axial, radial or dynamic load conditions.

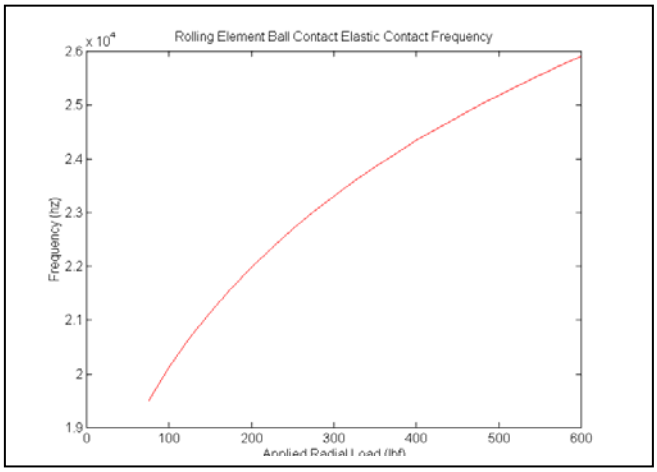


Figure 13. Ball Elastic Contact Frequency

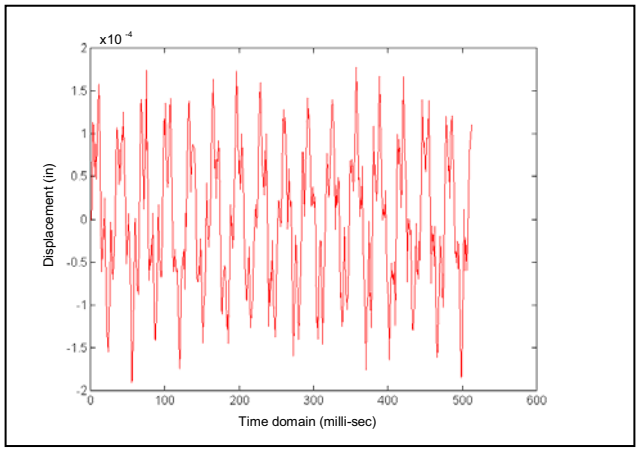


Figure 14. Outer Race y-Direction Displacement

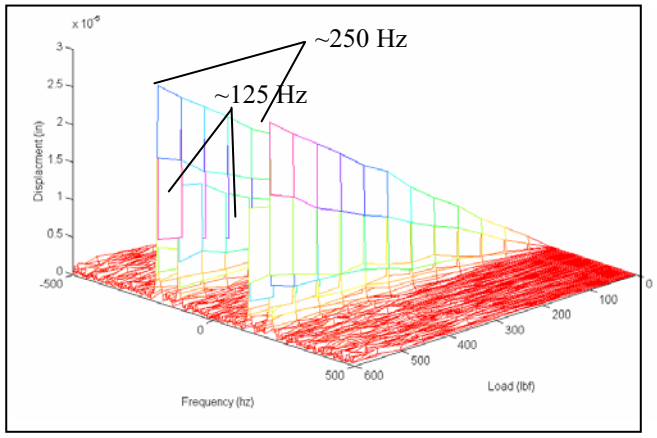


Figure 15. Outer Ring Vertical Displacement Frequency Response Function

The radial disks serve as dynamic loads and a source of imbalance loading, if required. The test bearing platform applies radial, axial, and misalignment loads to the test bearing outer ring. These data will be correlated with the physics-based finite element model results of each test bearing. The results of the experimental testbed may be verified with a kinematic-dynamic model of the bearing experiment testbed, which can replicate virtually any combination of load conditions and bearing faults.

Conclusion

We have primarily developed an accurate finite element of a bearing by considering detailed geometrical data as well as other kinematic constraints embedded in the bearing. The maximum ball deformation due to axial and radial load conditions appears to have somewhat a non-linear relationship when combined. Also, the FEM results show that the bearing is more sensitive to axial loads than radial loads in a linear manner. The primary results obtained from this research investigation are promising. Development of such a physics-based bearing model is also viable since bearing failure modes can be directly related with to operational stresses and material deformation levels.

Furthermore, the operational bearing acceleration and temperature signals can also be related to the internal bearing's response to the applied dynamic loads. The bearing stress and deformation levels, in a finite element model, can be carefully correlated with the response relationships to actual bearing degradation and failure modes. While the bearing failure detection and prevention is a complex task and application dependent, such response relationships could serve as a basis for a better bearing failure characterization.

Acknowledgement

This research was conducted as a part of the Office of Naval Research-MURI research initiative in cooperation with The Applied Research Laboratory (ARL) at Pennsylvania State University under contract number N00014-95-10461 and conducted in the Intelligent Manufacturing Research Laboratory at Tennessee State University. The authors gratefully acknowledge the support of ONR, The Applied Research Laboratory, and the various rolling bearing manufacturers who provided product catalogs and technical information.

References

1. Aktürk, N., Uneeb, M., Gohar, R., 1997, "The Effects of Number of Balls and Preload on Vibrations Associated with Ball Bearings", *Journal of Tribology*, Vol. 119, pp. 747-753.
2. American National Standards Institute, 1987, "Rolling Bearing Vibration and Noise (Method of Measuring)", *American National Standard (ANSI /AFBMA) Std 13-1987*.
3. Bhargava, V., Hahn, G. T., Rubin, C. A., 1985a, "An Elastic-Plastic Finite Element Model of Rolling Contact-Part 1: Analysis of Single Contacts", *Journal of Applied Mechanics*, Vol. 52, pp. 67-74.
4. Bhargava, V., Hahn, G. T., Rubin, C. A., 1985b, "An Elastic-Plastic Finite Element Model of Rolling Contact-Part 2: Analysis of Repeated Contacts", *Journal of Applied Mechanics*, Vol. 52, pp. 75-82.
5. Boston Gear Bearing Catalog, 1994.
6. Brauer, J. R., 1988, *What Every Engineer Should Know About Finite Element Analysis*, Marcel- Dekker, Inc, New York.
7. Dubeck, J., Shirkhodaie, A., and Garga, A., "Bearing Fault Diagnostics Using Physics-based Finite Element Modeling Technique", in the 53rd meeting of the Society for Machinery Failure Prevention Technology, April 21-24, Virginia Beach, VA, 1999.

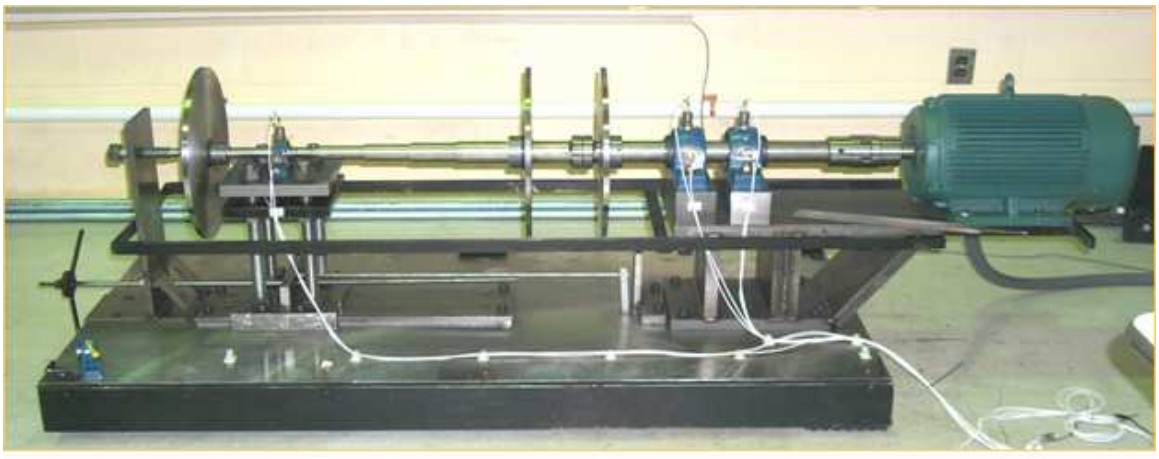


Figure 16. Tennessee State University Bearing Experimental Test bed.

8. Dubeck, J., Shirkhodaie, A., and Garga, A., "Physics-based Modeling and Diagnosis of High-Performance Bearings", in the Maintenance and Reliability Conference, MARCON'99, Gatlinburg, TN, May 10-12, 1999.

9. Hamrock, B. J., Jacobson, B. O., Schmid, S. R., 1999, *Fundamentals of machine elements*, WCB/McGraw-Hill, pp. 539-605.

10. Harris, T. A, 1966, *Rolling Bearing Analysis*, John Wiley & Sons, Inc, New York, pp. 36-59, 110-142.

11. Harris, T. A., McCool, J. I., 1996, "On the Accuracy of Rolling Bearing Fatigue Life Prediction", *Journal of Tribology*, Vol. 118, pp. 297-310.

12. Harris, T. A., 1997, "Prediction of Ball Fatigue Life in a Ball V-Ring Test Rig", *Journal of Tribology*, Vol. 119, pp. 365-374.

13. Hoeprich, M. R., 1992, "Rolling Element Bearing Fatigue Damage Propagation", *Journal of Tribology*, Vol. 114, pp. 328-333.

14. Kral, E. R., Komvopoulos, K., 1996, "Three-Dimensional Finite Element Analysis of Surface Deformation and Stresses in an Elastic-Plastic Layered Medium Subjected to Indentation and Sliding Contact Loading", *Journal of Applied Mechanics*, Vol. 63, pp. 365-375.

15. Lovell, M. R., Khonsari, M. M., Maragoni, R. D., 1997a, "Parameter Identification of Hysteresis Friction for Coated Ball Bearings Based on Three-Dimensional FEM Analysis", *Journal of Tribology*, Vol. 119, pp. 462-470.

16. Lovell, M. R., Khonsari, M. M., Maragoni, R. D., 1997b, "Frictional Analysis of MoS_2 Coated Ball Bearings: A Three-Dimensional FEM Analysis", *Journal of Tribology*, Vol. 119, pp. 754-763.

17. Meeks, C. R., Tran L., 1996, "Ball Bearing Dynamic Analysis Using Computer Methods-Part I: Analysis", ASME *Journal of Tribology*, Vol. 118, pp. 52-58.

18. MSC/ARIES Reference Manual, 1995, MacNeal-Schwendler Corporation, Los Angeles, CA.

19. MSC/NASTRAN Reference Manual, 1996, MacNeal-Schwendler Corporation, Los Angeles, CA.

20. Shirkhodaie, A., 1997, "Predictive and Preventive Maintenance of Mechanical Systems Using Neural Networks and Fuzzy Logic Technique," The 1st Progress Report Submitted to The Applied Research Laboratory at Pennsylvania State University, State College, PA, 1997.

21. Shirkhodaie, A., Subramanian, S., Muthuswamy, K., 1998a "A Hybrid Neural Network-Fuzzy Logic Fault Diagnostic Technique For Bearings Condition-Based Maintenance" submitted for presentation in the SAE Advances in Aviation Safety Conference and Exposition, April 6-8, 1998, Daytona Beach, FL.

22. Shirkhodaie, A., 1998b, "Advances on Health Monitoring and Faults Diagnosis and Prognosis of Machinery," in The Sixth Annual Mechanical Engineering Conference of Iran Society of Mechanical Engineers, Tehran, Iran, May 10-15, 1998.

23. Shirkhodaie, A., Dubeck, J., Garga, A., "Physics-based Modeling and Diagnosis of High Performance Bearings" will appear in the MARCON '99 conference, May 10-12, 1999, Gatlinburg, TN. 24. Shigley, J. E., Mischke, C. R., 1983, *Mechanical Engineering Design*, McGraw-Hill, New York, NY, pp. 273-340.

25. SKF General Catalog *4000 US*, 1991, A. B. SKF, Gothenburg, Sweden, pp. 25-42.

26. Yhland, E., 1992, "A Linear Theory of Vibrations Caused by Ball Bearings With Form Errors Operating at Moderate Speed", ASME *Journal of Tribology*, Vol. 114, pp. 348-359.

27. Zaretsky, E.V., Poplawski, J. V., Peters, S. M., 1996, "Comparison of Life Theories for Rolling Element Bearings", *Tribology Transactions*, Vol. 39, pp. 237-248.

28. Zhao, H, 1998, "Analysis of Load Distributions Within Solid and Hollow Roller Bearings", *Journal of Tribology*, Vol. 120, pp. 134-139.

Dynamic Multiple Fault Diagnosis: Mathematical Formulations and Solution Techniques

Satnam Singh, Kihoon Choi, Anuradha Kodali and Krishna Pattipati
Department of Electrical and Computer Engineering,
University of Connecticut, 371 Fairfield Road, U-2157,
Storrs, CT 06269, USA
{krishna, satnam}@engr.uconn.edu
John W. Sheppard
Department of Computer Science,
The Johns Hopkins University,
3400 N. Charles Street,
Baltimore, MD 21218, USA
Setu Madhavi Namburu, Shunsuke Chigusa, Danil V. Prokhorov and Liu Qiao
Toyota Technical Center USA
1555 Woodridge, RR #7
Ann Arbor, MI 48105, USA

Abstract

Dynamic multiple fault diagnosis (DMFD) is a challenging and difficult problem due to coupling effects of the states of components and imperfect test outcomes that manifest themselves as missed detections and false alarms. The objective of the DMFD problem is to determine the most likely temporal evolution of fault states, the one that best explains the observed test outcomes over time.

Here, we discuss four formulations of the DMFD problem. These include the deterministic situation corresponding to a perfectly-observed coupled Markov decision processes, to several partially-observed factorial hidden Markov models ranging from the case where the imperfect test outcomes are functions of tests only to the case where the test outcomes are functions of faults and tests, as well as the case where the false alarms are associated with the nominal (fault-free) case only. All these formulations are intractable NP-hard combinatorial optimization problems. We solve each of the DMFD problems by decomposing them into separable subproblems, one for each component state sequence.

Our solution scheme can be viewed as a two-level coordinated solution framework for the DMFD problem. At the top (coordination) level, we update the Lagrange multipliers (coordination variables, dual variables) using the subgradient method. The top level facilitates coordination among each of the subproblems, and can thus reside in a vehicle-level diagnostic control unit. At the bottom level, we use a dynamic programming technique (specifically, the Viterbi decoding or Max-sum algorithm) to solve each of the subproblems. The key advantage of our approach is that it provides an approximate duality gap, which is a measure of suboptimality of the DMFD solution. Interestingly, the perfectly-observed DMFD problem leads to a dynamic set covering problem, which can be approximately solved via Lagrangian relaxation and Viterbi decoding. Computational results on real-world problems are presented.

Introduction

Safety critical systems, such as aircraft, automobiles, nuclear power plants and space vehicles, are becoming significantly more complex and interconnected. The recent advances in wireless technology, remote communication, computational capabilities, sensor technology and standardized hardware/software interfaces have further increased the complexity of these systems. This complexity may result in failures of multiple components. Hence, there is a need to develop smart on-board diagnostic algorithms that can determine the most likely set of failure causes in a system, given observed test outcomes over time.

The multiple fault diagnosis (MFD) problem originates in several fields such as medical diagnosis [1], error correcting codes, speech recognition, distributed computer systems and networks [2]. The MFD problem in large-scale systems with unreliable tests was first considered by Shakeri et al. in [3]. They proposed near-optimal algorithms using Lagrangian relaxation and subgradient optimization methods for the static MFD problem. In the area of distributed system management, the MFD problem is studied by Odintsova et al. in [2]. They utilized an adaptive diagnostic technique, termed active probing, for fault diagnosis and isolation. A probe can be viewed as a test in our terminology; the purpose of a probe is to check the set of system components on the probed path. The probe outcomes determine if one or more of the components on the probed path are faulty or normal. Given the probe outcomes, a diagnostic matrix (D-matrix, diagnostic dictionary, reachability matrix) defining the relationship among the probes and component faults, and the initial system state, they developed a sequential multi-fault algorithm to diagnose the system state. They considered the probe outcomes as being deterministic,

which is analogous to the assumptions made in our Problem 4, and in the work described in [11]-[14]. In [4], Le et al. applied graphical model-based decoding algorithms to the MFD problem in the presence of unreliable tests. They proposed a suboptimal belief propagation algorithm used to decode low density parity check codes. They considered a fault model, where tests are asymmetric, i.e., the D-matrix is not binary and the test outcomes are also unreliable, and they termed it the Y model. Their implementation is parallel to our Problem formulation 1; however, they considered only the static case.

The DMFD problem refers to determining the most likely temporal evolution of component states, given a set of partial and unreliable test outcomes over time. The dynamic single fault diagnosis problem using a hidden Markov model (HMM) formalism was first proposed by Ying et al. [5], where it is assumed that, at any time, the system has at most one fault state present. This modeling is somewhat unrealistic for most real-world systems. Another version of the dynamic fault diagnosis problem was studied in [6]: unknown probabilities of sensor error, incompletely-populated sensor observations, and multiple faults were allowed, but the faults could only occur or clear once per sampling interval. Another approach, developed by Ruan et al. [7], decomposes the original DMFD problem into a series of decoupled subproblems, one for each epoch. For a single epoch MFD, they developed a deterministic simulated annealing (DSA) method, which is inspired by its sibling stochastic simulated annealing and the approximate belief revision (ABR) heuristic algorithm [1]. This algorithm enlarges the search space of ABR via DSA, and is guaranteed to provide a solution no worse than (often significantly better) than ABR. The single epoch MFD was extended to incorporate fault states of multiple consecutive epochs. In addition, they applied a local search and update scheme to further smooth the "noisy" diagnoses stemming from imperfect test results and, thereby, increase the accuracy of fault diagnosis.

The DMFD problem can also be viewed as a factorial HMM (FHMM), a simplified Markovian dynamic Bayesian network, discussed in the machine learning literature [8]. Here, the HMM state is factored into multiple state variables, and is represented in a distributed manner. The authors in [8] discussed an exact algorithm for inference in FHMM. Here, the inference and learning involves computing the posterior probabilities of multiple hidden layers (or states), given the test outcomes. However, due to the combinatorial nature of the hidden state representation, the exact algorithm is intractable. They presented approximate inference algorithms based on Gibbs sampling and variational methods. The latter methods are similar to Lagrangian relaxation, although motivated from a Fenchel duality perspective [1], [17].

In our recent work [9], we extended the work of Ruan et al. [7], Shakeri et al. [3] and Tu et al. [11] on MFD to solve the DMFD problem by combining the Viterbi algorithm and Lagrangian relaxation in an iterative way. This paper is an extension of our work in [9]. Depending on the probabilistic assumptions on fault-test relationships and test outcomes, one obtains various DMFD formulations. In [9], we discussed only DMFD formulation 1 and it was solved only for small-scale systems. In this paper, we provide three other formulations of the DMFD problem along with their solutions. Here, we also compare the results between the subgradient and the deterministic simulated annealing methods [7]. Simulation results on several real world systems are provided for our earlier formulation of the DMFD problem (formulation 1).

DMFD Problem Formulations

The dynamic multiple fault diagnosis problem consists of a set of possible fault states in a system, and a set of binary test outcomes that are observed at each sample (observation, decision) epoch. Fault states are assumed to be independent. Each test outcome provides information on a subset of the fault states. At each sample epoch, a subset of test outcomes is available. Tests are imperfect in the sense that the outcomes of some of the tests could be missing, and tests have missed-detection/false-alarm processes associated with them. The observations consist of imperfect binary test outcomes, and are characterized by sets of passed tests outcomes, O_p and failed tests outcomes, O_f. Formally, we represent the DMFD problem as $DM = \{S, \kappa, T, O, D, P, A\}$, where $S = \{s_1, ..., s_m\}$ is a finite set of *m* components (failure sources) associated with the system. The state of component s_i is denoted by $x_i(k)$ at epoch *k*, where $x_i(k) = 1$ if failure source s_i is present; $x_i(k) = 0$, otherwise. Here, $\kappa = \{0, 1, ..., k, ...K\}$ is the set of discretized observation epochs. The status of all component states at epoch *k* is denoted by $\underline{x}(k) = \{x_1(k), x_2(k), ..., x_m(k)\}$. We assume that the initial state $\underline{x}(0)$ is known (or its probability distribution is known).

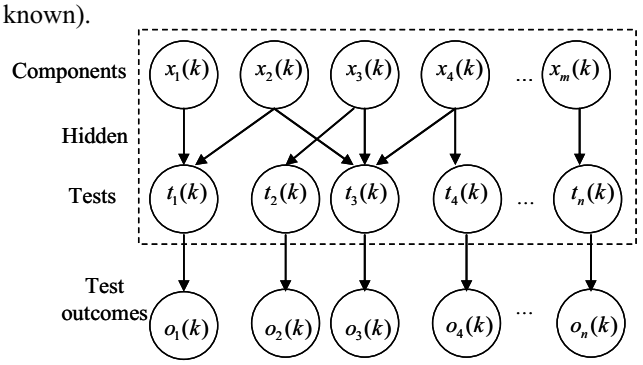


Figure 1: Tri-partite digraph for DMFD problem

The observations at each epoch are subsets of binary outcomes of tests $O = \{o_1, o_2, ..., o_n\}$, i.e., $o_j \in \{pass, fail\} = \{0,1\}$. Figure 1 shows the DMFD problem as a tri-partite digraph at epoch *k*. Component states, tests and test outcomes represent the nodes of the digraph. Here, the true states of the component states and tests are hidden. $P = \{Pd, Pf\}$ represents a set of probabilities of detection and false alarm, which is defined differently for each of the DMFD problem formulations. We also define the matrix $D = [d_{ij}]$ as the dependency matrix (D-matrix), which represents the full-order dependency among failure sources and tests.

Each component state is modeled as a two-state non-homogenous Markov chain. For each component state, e.g., for component s_i at epoch *k*, $A = (Pa_i(k), Pv_i(k))$ denotes the set of fault appearance probability $Pa_i(k)$ and fault disappearance probability $Pv_i(k)$ defined as $Pa_i(k) = \Pr(x_i(k) = 1 \mid x_i(k-1) = 0)$ and $Pv_i(k) = \Pr(x_i(k) = 0 \mid x_i(k-1) = 1)$. These probabilities are required to model the intermittent faults. Here, $T = \{t_1, t_2, ..., t_n\}$ is a finite set of *n* available binary tests, where the integrity of the system can be ascertained. We denote the set of passed tests, T_p and failed tests T_f. At each observation epoch, *k*, $k \in \kappa$, test outcomes upto and including epoch *k* are available, i.e., we let $O^k = \{O(b) = (O_p(b), O_f(b))\}_{b=1}^{k}$, where O^k is the set of observed test outcomes at epoch *k*, with $O_p(b)(\subseteq O)$ and $O_f(b)(\subseteq O)$ as the sets of passed and failed tests at epoch b, respectively. The tests are partially observed in the sense that outcomes of some tests may not be available, i.e., $(O_p(b) \bigcup O_f(b)) \subset O$. In addition, tests exhibit missed detections and false alarms. Here, we also make the noisy-OR ("causal independence") assumption [10].
The DMFD problem can be formulated in the following ways, arranged from the general to simplified:

Problem 1: When the probability of detection (Pd_{ij}) and false alarm probability (Pf_{ij}) are associated with each test and each fault class, i.e., $Pd_{ij} = \Pr(o_j(k) = 1 \mid x_i(k) = 1)$ and $Pf_{ij} = \Pr(o_j(k) = 1 \mid x_i(k) = 0)$ of a failure source s_i and test t_j. For notational convenience, when s_i does not affect the outcome of test t_j, we let the corresponding $Pd_{ij} = Pf_{ij} = 0$. This problem scenario frequently arises in medical fault diagnosis. For example, the QMR-DT (Quick Medical Reference, Decision-Theoretic) database used in the domain of internal medicine, contains approximately 600 disease nodes (faults or failure sources) and 4000 symptoms (tests) [7]. Each of the symptoms could have a probability pair (Pd_{ij}, Pf_{ij}) associated with them. Figure 2 shows the bi-partite graph, where the edges represent the probability pair (Pd_{ij}, Pf_{ij}). These probabilities can be obtained from the tri-partite digraph (Figure 1) using the total probability theorem as follows:

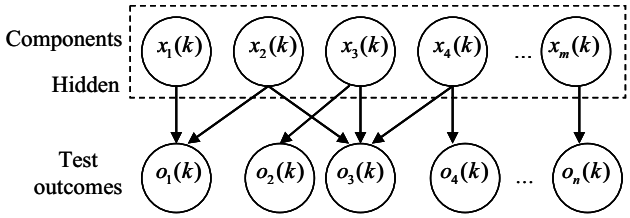


Figure 2: Bi-partite graph for DMFD problem 1 and 2

$$\Pr(o_j(k) \mid x_i(k)) = \sum_{t_j \in \{0,1\}} \Pr(o_j(k), t_j(k) \mid x_i(k)) = \sum_{t_j \in \{0,1\}} \Pr(o_j(k) \mid t_j(k)) \Pr(t_j(k) \mid x_i(k)) \quad (1)$$

Problem 2: In situation where the probability of detection (Pd_{ij}) is associated with each failure source-test pair, but the false alarm probability is specified only for the normal system state, i.e., $Pf_j = P(o_j(k) = 1 \mid x_1(k) = 0, ..., x_m(k) = 0)$, we obtain a slightly complicated variation of Problem formulation 1 (in terms of computational complexity, but not in terms of parameterization). This type of scenario arises when we design class-specific classifiers that distinguish between normal system operation and failure source, s_i only, or when the false alarms are defined on an overall system basis. Here, the probability pair (Pd_{ij}, Pf_j) is associated with test outcomes to model imperfect test outcomes [3]. This model is also called the Z model in [4]. Similar to problem 1, the probability pair (Pd_{ij}, Pf_j) is shown as edges between the hidden component states and test outcomes in Figure 2, and they can be obtained from the tri-partite digraph (Figure 1) using the total probability theorem on the nodes of test layer.

Problem 3: When the probability of detection (Pd_j) and false alarm probability (Pf_j) are associated with each test t_j only. The probability pair (Pd_j, Pf_j) is shown as the edges between the tests and test outcomes in the tri-partite digraph (Figure 1). This formulation is quite useful in classifier fusion using error correcting codes. In the error correcting code (ECC) matrix, each column corresponds to a binary classifier with the associated (Pd_j, Pf_j) pair, which are learned during training and validation. This type of formulation is also considered in [6].

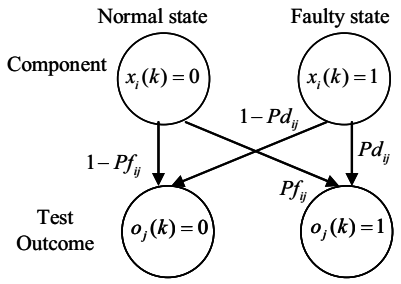


Figure 3: Detection and false alarm probabilities for problem 1

This formulation provides a nice vehicle for the dynamic fusion of classifiers, where each column of the ECC matrix is a classifier, and their associated probability pairs (Pd_j, Pf_j) are uncertainties associated with classifier outcomes. When the learned parameters and the ECC matrix are fed as an input to the DMFD algorithm, it performs dynamic fusion of classifier outputs over time. Note that the sampling interval of the dynamic fusion algorithm can be different from the sampling interval of the raw sensor data.

Problem 4: This is the deterministic case when tests are perfect i.e. $Pd_{ij}=1$ and $Pf_{ij}=0$ [11]. This formulation reduces the tripartite digraph in Figure 1 to a bipartite graph between the components and tests. This scenario is useful in situations where the tests are highly reliable (e.g., automated testing of electronic cards), and leads to a novel dynamic set covering problem.
Next, we discuss the DMFD formulations in detail.

DMFD Problem 1

In this problem, we assume that the detection and false alarm probabilities (Pd_{ij}, Pf_{ij}) are associated with each failure source and each test. Figure 3 illustrates these probabilities. We have presented this problem in detail in [9]. Here, we revise only the key steps of the problem and solution.

The DMFD problem is one of finding, at each decision epoch *k*, the most likely fault state candidates $\underline{x}(k)\in\{0,1\}^m$, i.e., the fault state evolution over time, $X^K=\{\underline{x}(1),...,\underline{x}(K)\}$, that best explains the observed test outcome sequence O^K. We formulate this as one of finding the maximum *a posteriori* (MAP) configuration:

$$\widehat{X}^K=\arg\max_{X^K}\Pr(X^K\mid O^K) \tag{2}$$

In [9], we showed that we obtain optimal fault sequence $\widehat{X}^K$ using the following primal problem:

$$\widehat{X}^K=\arg\max_{X^K,Y^K}J(X,Y)=\arg\max_{X^K,Y^K}\sum_{k=1}^{K}f_k(\underline{x}(k),\underline{x}(k-1),\underline{y}(k)) \tag{3}$$

where the fault state sequence is $X^K=\{\underline{x}(1),\underline{x}(2),...,\underline{x}(K)\}$ and $Y^K=\{\underline{y}(1),\underline{y}(2),...,\underline{y}(K)\}$ $\underline{y}(k)=\{y_j(k),\forall j\in O_f(k)\}$ are new variables such that

$$\ln y_j(k)=\sum_{i=1}^{m}c_{ij}x_i(k)+\eta_j,\quad \forall j\in O_f(k). \tag{4}$$

Here, the primal objective function for an individual fault state, i.e., $f_k(\underline{x}(k),\underline{x}(k-1),\underline{y}(k))$ is defined as

$$f_k(\underline{x}(k),\underline{x}(k-1),\underline{y}(k))=\sum_{o_j\in O_p(k)}\sum_{i=1}^{m}c_{ij}x_i(k)+\sum_{i=1}^{m}\mu_i(k)x_i(k)$$
$$+\sum_{o_j\in O_f(k)}\ln(1-y_j(k))+\sum_{i=1}^{m}\sigma_i(k)x_i(k-1)$$
$$+\sum_{i=1}^{m}h_i(k)x_i(k)x_i(k-1)+\gamma(k)+g(k) \tag{5}$$

where, the parameters c_{ij}, $\gamma(k)$, η_j are functions of Pd_{ij} and Pf_{ij} and $\mu_i(k)$, $\sigma_i(k)$, $h_i(k)$, $g(k)$ are functions of $Pa_i(k)$, $Pv_i(k)$. Note that the multiple HMMs are coupled here because their states are observed only via a set of test outcomes. In equation (5), the terms involving $y_j(k)$ and $h_i(k)$ shows the coupling effects. The detailed steps of deriving the primal problem are provided in [9].

The primal DMFD problem posed in (3)-(5) is NP-hard which, for all practical purposes, means that, unless P=NP, it cannot be solved to optimality within a polynomially bounded computation time. The NP-hard nature of the primal DMFD problem motivates us to decompose it into a primal-dual problem using a Lagrangian relaxation approach. By defining new variables and constraints, the DMFD problem reduces to a combinatorial optimization problem with a set of equality constraints. The constraints are relaxed via Lagrange multipliers.

In [9], we also showed that the dual problem of the primal DMFD problem as posed in (3)-(5), can be written as

$$\min_{\Lambda}\quad Q(\Lambda)$$
$$\text{subject to } \Lambda=\{\lambda_j(k)\ge 0,k\in(1,K),j\in O_f(k)\} \tag{6}$$

where the dual function $Q(\Lambda)$ is defined by

$$Q(\Lambda)=\max_{X^K}\sum_{i=1}^{m}Q_i(\Lambda). \tag{7}$$

Here

$$Q_i(\Lambda)=\sum_{k=1}^{K}\xi_i(x_i(k),x_i(k-1),\lambda_j(k))+\frac{1}{m}w_k(\Lambda) \tag{8}$$

$$\xi_i(x_i(k), x_i(k-1), \lambda_j(k)) =$$
$$\left(\sum_{o_j \in O_p(k)} c_{ij} + \mu_i(k) - \sum_{o_j \in O_f(k)} c_{ij}\lambda_j(k) \right) x_i(k)$$
$$+\sigma_i(k)x_i(k-1) + h_i(k)x_i(k)x_i(k-1) \quad (9)$$

and

$$w_k(\Lambda) = \gamma(k) + g(k)$$
$$+ \sum_{\forall o_j \in O_f(k)} \left[\lambda_j(k)\ln\lambda_j(k) - (1+\lambda_j(k))\ln(1+\lambda_j(k)) - \lambda_j(k)\eta_j \right] \quad (10)$$

represents the dual function for the i^{th} component. The main benefit of (7) is that now the original problem is separable. Using the Lagrangian relaxation method, we decomposed the original DMFD problem into *m* separable subproblems, one for each component state sequence $\underline{x}_i$, where $\underline{x}_i = \{x_i(1), x_i(2), ..., x_i(K)\}$, $x_i(k) \in \{0,1\}$ and $i \in \{1, m\}$. The relaxation procedure generates an upper bound for the primal objective function. The procedure of minimizing the upper bound via a subgradient subgradient optimization produces a sequence of dual feasible, and the concomitant primal feasible solutions to the DMFD problem. If the objective function value for the best feasible solution and the upper bound are the same, the feasible solution is the optimal solution. Otherwise, the difference between the upper bound and the feasible solution, termed the approximate duality gap, provides a measure of suboptimality of the DMFD solution; this is a key advantage of our approach. Details of the DMFD algorithm, subgradient method and dynamic programming are provided in [9].

In this paper, we also compared results of subgradient method with deterministic simulated annealing method [7] in the results section.

DMFD Problem 2

In this formulation, we define Pd_{ij} as $Pd_{ij} = \Pr(o_j(k) = 1 \mid x_i(k) = 1)$ and $Pf_j = \Pr(o_j(k) = 1 \mid x_1(k) = 0, x_2(k) = 0, ..., x_m(k) = 0)$. This scenario is depicted in Figure 4.

Here, the DMFD problem is equivalent to

$$\hat{X}^K = \arg\max_{X^K} J(\underline{x}(k), \underline{x}(k-1)) = \arg\max_{X^K} \sum_{k=1}^{K} f_k(\underline{x}(k), \underline{x}(k-1)) \quad (11)$$

where the primal objective function for an individual component state, i.e., f_k is defined as

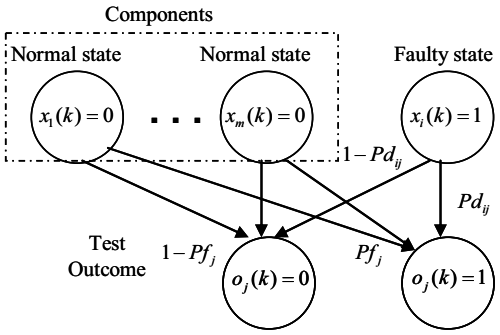


Figure 4: Detection and false alarm probabilities for problem 2

$$f_k(\underline{x}(k), \underline{x}(k-1), \underline{y}(k), z(k)) = z(k)\eta(k) + g(k)$$
$$+\sum_{i=1}^{m} \sum_{o_j(k) \in O_p(k)} x_i(k)\ln(1 - Pd_{ij}) + \sum_{o_j(k) \in O_f(k)} \ln(1 - y_j(k))$$
$$+\sum_{i=1}^{m} \tau_i(k)x_i(k) + \sum_{i=1}^{m} \sigma_i(k)x_i(k-1) + \sum_{i=1}^{m} h_i(k)x_i(k)x_i(k-1) \quad (12)$$

where

$$\ln z(k) = \sum_{i=1}^{m} \ln(1 - x_i(k)), \quad (13)$$

$$\ln(y_j(k)) = z(k)\ln(1 - Pf_j) + \sum_{i=1}^{m} x_i(k)\ln(1 - Pd_{ij}), \quad (14)$$

$\eta(k)$ is a function of Pf_j and $\tau_i(k)$, $\sigma_i(k)$, $h_i(k)$, $g(k)$ are functions of $Pa_i(k)$, $Pv_i(k)$. Appending constraints (13) and (14) via Lagrange multipliers $\mu(k)$, $\{\lambda_j(k)\}_{j \in O_f(k)}$, the Lagrangian function $L(X, Y, z, \Lambda)$ can be obtained. Using the Lagrange multiplier theorem, we optimize the Lagrangian function $L(X, Y, z, \Lambda)$ w.r.t. $y_j(k)$ to obtain optimal $y_j(k)^*$ and optimizing w.r.t. $z(k)$, we obtain optimal $z(k)^*$. The dual function $Q(\Lambda)$ of problem 2 is defined by

$$Q(\Lambda) = \max_{X^K, Y^K, z^K} L(X, Y, z, \Lambda). \quad (15)$$

Substituting $(y_j(k)^*, z(k)^*)$ into $L(X, Y, z, \Lambda)$ and simplifying further by rearranging and combining the terms, we obtain the dual function as

$$Q(\Lambda) = \max_{X^K} \sum_{i=1}^{m} Q_i(\Lambda) \quad (16)$$

where

$$Q_i(\Lambda) = \sum_{k=1}^{K} \xi_i(x_i(k), x_i(k-1), \lambda_j(k), \mu(k))$$
$$+ \frac{1}{m} w_k(\lambda_j(k), \mu(k)) \quad (17)$$

and

$$\xi_i(x_i(k), x_i(k-1), \lambda_j(k), \mu(k))$$
$$= \left(\sum_{o_j(k) \in O_p(k)} \ln(1-Pd_{ij}) + \tau_i(k) - \sum_{o_j(k) \in O_f(k)} \lambda_j(k) \ln(1-Pd_{ij}) \right) x_i(k)$$
$$+ \sigma_i(k) x_i(k-1) + h_i(k) x_i(k) x_i(k-1) - \mu(k) \ln(1-x_i(k)) \quad (18)$$

and

$$w_k(\lambda_j(k), \mu(k)) = \mu(k) \left(\frac{\eta_j(k) + \ln(\mu(k))}{-\eta_j(k) + \sum_{\forall o_j \in O_f(k)} \lambda_j(k) \ln(1-Pf_j)} \right)$$
$$- \mu(k) \left(\sum_{\forall o_j \in O_f(k)} \frac{\lambda_j(k) \ln(1-Pf_j)}{-\eta_j(k) + \sum_{\forall o_j \in O_f(k)} \lambda_j(k) \ln(1-Pf_j)} \right)$$
$$+ g(k) + \sum_{\forall o_j \in O_f(k)} \left[\lambda_j(k) \ln \lambda_j(k) - (1+\lambda_j(k)) \ln(1+\lambda_j(k)) \right] \quad (19)$$

The dual problem posed in (15)-(19) is separable and it can be solved by following a procedure similar to that used for solving Problem 1. The only difference is that we also need to update the Lagrange multiplier $\mu(k)$ using a subgradient method.

DMFD Problem 3

In this formulation, we consider the case where the probabilities of detection and false alarm (Pd_j, Pf_j) are associated only with each test t_j (see Figure 5). Formally, $Pd_j = \Pr(o_j(k)=1 \mid t_j(k)=1)$ and $Pf_j = \Pr(o_j(k)=1 \mid t_j(k)=0)$. We can convert these probabilities into a special case of Problem Formulation 1 by computing (Pd_{ij}, Pf_{ij}) using (1):

$$Pd_{ij} = (d_{ij}) Pd_j + (1-d_{ij}) Pf_j \quad (20)$$

Similarly

$$Pf_{ij} = (d_{ij}) Pf_j + (1-d_{ij}) Pd_j \quad (21)$$

The solution of Problem 3 can be obtained by substituting Pd_{ij} and Pf_{ij} in (20)-(21) in the solution of Problem 1.

DMFD Problem 4

Next, we consider the case when the system consists of reliable tests, and the fault-test relationships are deterministic, i.e. $Pd_{ij} = 1$ and $Pf_{ij} = 0$ for $i = 1,...,m$ and $j = 1,...,n$ or equivalently, the D-matrix completely characterizes the fault-test relationships [11]. This formulation can be represented as a bipartite graph between the components and tests. In this case, if some tests have passed, then we can infer that all the failure sources covered by these tests are good components.

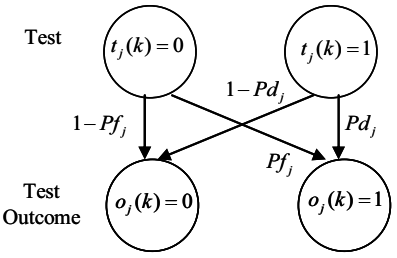


Figure 5: Detection and false alarm probabilities for problem 3

Thus, we need to infer failed components from those covered by the failed tests only, i.e., by excluding those components covered by the passed tests. Consequently, the size of the DMFD problem can be reduced by removing all failure sources $\{s_i \mid Pf_{ik}=0, Pd_{ik}=1, \text{ and } t_k \in T_p(k)\}$. For each failed test $t_j(k) \in T_f(k)$, the optimal solution contains at least one component state $x_i(k) = 1$ that satisfies $d_{ij} = 1$. Thus, there must be one or more failure sources that cover the failed tests. Let us consider a matrix A, which has each row representing the list of failure sources covered by a failed test. After excluding the failure sources covered by the passed tests, the resulting matrix A is a binary matrix such that $a_{ij} = d_{ji}$. After substituting $Pd_{ij} = 1$ and $Pf_{ij} = 0$ in (5), the reliable test scenario with a binary D-matrix simplifies to a dynamic set covering problem with the following objective function term at epoch *k*:

$$f_k(\underline{x}(k), \underline{x}(k-1)) = \sum_{i=1}^{m} \mu_i(k) x_i(k) + \sum_{i=1}^{m} \sigma_i(k) x_i(k-1)$$
$$+ \sum_{i=1}^{m} h_i(k) x_i(k) x_i(k-1) + g(k) \quad (22)$$

subject to following constraints:

$$A(k)\underline{x}(k) \geq \underline{e} \text{ for } t_j(k) \in T_f(k) \quad (23)$$

where $\underline{e}$ is a vector of one's. Appending constraints (23) to (22) via Lagrange multipliers $\{\lambda_j(k)\}_{j \in T_f(k)}$, the Lagrangian function $L(X, \Lambda)$ can be obtained. The dual function $Q(\Lambda)$ is defined by

$$Q(\Lambda) = \max_{X^K} L(X, \Lambda). \quad (24)$$

Simplifying further by rearranging and combining the terms, we obtain the dual function as

$$Q(\Lambda) = \max_{X^K} \sum_{i=1}^{m} Q_i(\Lambda) \quad (25)$$

where

$$Q_i(\Lambda) = \sum_{k=1}^{K} \xi_i(x_i(k), x_i(k-1), \lambda_j(k)) + \frac{1}{m} w_k(\Lambda) \quad (26)$$

$$\xi_i(x_i(k), x_i(k-1), \lambda_j(k)) = - \sum_{t_j \in T_f(k)} \lambda_j(k) a_{ji}(k) x_i(k)$$
$$+ \mu_i(k) x_i(k) + \sigma_i(k) x_i(k-1) + h_i(k) x_i(k) x_i(k-1) \quad (27)$$

and

$$w_k(\Lambda) = g(k) + \sum_{\forall t_j \in T_f(k)} \lambda_j(k) . \quad (28)$$

The dual problem defined in (24)-(28) is separable. The Viterbi algorithm is used to solve each subproblem corresponding to each fault state sequence $\underline{x}_i$. This algorithm can be viewed as a dynamic set covering problem, which is NP-hard. Thus, the dynamic set covering problem is solved by combining the Viterbi algorithm and Lagrangian relaxation. This generalizes Beasley's Lagrangian relaxation algorithm for the static set covering problem [11], [15] to dynamic settings.

Results

We implemented and applied the solution of problem 1, the most general version of the DMFD problem formulation, to a few real world models. Table 4 illustrates the model parameters of an automotive system, a document matching system (Docmatch), a power distribution system (Powerdist), a UH-60 helicopter transmission system (Helitrans) and an engine simulator (EngineSim). Details of these models are provided in [11]. Here, *m*, *n*, and *c* denote the number of components (failure sources), number of tests and the average number of intermittent faults that can occur over a span of 100 epochs. The fault appearance probabilities (Pa_i) were computed based on the average number of intermittent faults (*c*). These real-world systems are not ideal because they have fewer tests as compared to failure sources; hence, some failure sources are not covered by any tests. The fault disappearance probabilities (Pv_i) were varied between 0.0025-0.0049 to allow *c* intermittent faults, on average. The probabilities of detection and false alarm were varied as shown in Table 4. The maximum number of subgradient iterations was set at 80. The algorithms were implemented in MATLAB. We used a standard PC having Pentium 4 Processor with 3.0 GHz clock speed and 512 MB RAM.

Table 5 shows the results obtained using the subgradient (S) and the deterministic simulated annealing (DSA) [7] methods. Here, J, Q, D, CI, FI and t denote the primal function value, the dual function value, the approximate duality gap, the correct isolation rate, the false alarm rate and the computation time per epoch. The primal and dual function values are computed using (3)-(5) and (15)-(19), respectively. The approximate duality gap (*D*) is computed as a ratio of the difference between *Q* and *J* divided by the absolute value of the primal feasible value *J*. Here, *CI* is computed as the average percentage of true fault states, which are isolated by the algorithm over a span of *K* epochs, and *FI* is computed as the average percentage of fault states, which are falsely isolated by the algorithm over a span of *K* epochs. The subgradient method (S) achieves higher correct isolation rates as compared to (DSA) for all the systems except Helitrans. However, the DSA method achieves better primal function value and is also effective in reducing the computation time (*t*). Also, note that we can obtain a hybrid duality gap by taking the maximum primal solution from the subgradient (S) and the deterministic simulated annealing (DSA) methods and the dual function value from the subgradient (S) method. The hybrid DSA-subgradient (HS) duality gaps are also shown in Table 6. The CPU time (t) is measured in seconds. Based on our experience, these numbers are highly practical and they can be further reduced by a factor of 10 when implemented in the C language.

We also showed an application of the DMFD Problem 3 formulation in our recent paper [16] where we performed dynamic fusion of classifiers over time for automotive engine fault diagnosis. The temporal correlations considered by dynamic fusion improve classification accuracy over a variety of static fusion techniques (based on batch data).

Table 4: Real world models

	m	*n*	Pd_{ij} Pf_{ij}	*c*, Pa_i
Automotive	22	60	(0.85-0.95), 0-0.02	3, 9.13e-04
Docmatch	257	180	(0.6-1),0	9, 3.12e-04
Powerdist	96	98	(0.6-1),0	3,3.13e-04
Helitrans	34	51	(0.6-1),0	2, 2.95e-04
EngineSim	53	30	(0.6-1),0	2,5.68e-04

Table 5: Results on real world models

		J	*Q*	*D* (%)	*CI*	*FI*	*t*
Automotive	S	-658	-481	27	99.5	0.05	0.43
	DSA	-775	--	--	75	1.30	0.01
	HS	-658	-481	27			
Docmatch	S	-541	-311	42.5	88.2	0.36	4.53
	DSA	-405	--	--	69	0.70	0.24
	HS	-405	-311	23.2			
Powerdist	S	-232	-125	46.1	91.6	0.75	1.56
	DSA	-157	--	--	84	0.30	0.05
	HS	-157	-125	20.3			
Helitrans	S	-15	-14	6.7	94.8	0.31	0.47
	DSA	-15	--	--	100	0.0	0.02
	HS	-15	-14	6.7			
EngineSim	S	-85	-33	61.1	95.1	2.28	0.64
	DSA	-51	--	--	86	0.3	0.02
	HS	-51	-33	35.3			

Complexity: All the DMFD formulations are NP-hard. The simplest one, i.e. the deterministic formulation (Problem 4) is also NP-hard. If we use brute force dynamic programming (DP), the complexity is $O(K2^{2m})$ where K is total number of epochs and m is number of components (failure sources). The DP is infeasible for systems with more than about 15 states. The algorithm presented here reduces the overall complexity to $O(K(m+O_f))$ where O_f is the set of failed tests outcomes, a substantial improvement. In particular, the complexities of binary Viterbi algorithm over all fault states and the subgradient method are $O(Km)$ and $O(KO_f)$, respectively, per iteration.

Conclusions

This paper discussed four formulations of the DMFD problem. Analogous forms of these formulations have been studied widely in fault diagnosis community in a static context, and applied in various fields. Here, we provided a unified formulation of all the MFD formulations in a dynamic context. The first formulation refers to a generalized version of the DMFD problem when the detection and false alarms probabilities are associated with each test and fault. In the second formulation, the false alarm probability is associated with fault-free case only. The solution to the second formulation was shown to be quite similar to that of problem formulation 1, except for the need to update an additional Lagrange multiplier. The third formulation considers the case where the uncertainties are associated with only test outcomes. This models dynamic fusion of classifier outputs. In the fourth formulation, we considered the deterministic case, which led to a novel dynamic set covering problem.

We simulated problem 1 using real world models. Results demonstrate that our algorithm achieves high isolation rate as compared the deterministic simulated annealing method. The latter provides better primal function value as compared to the subgradient method. In our future work, we plan to implement an on-line version of our algorithm using a sliding window method. The sliding window implementation will reduce the number of Lagrange multipliers updates, because estimates for these would have been computed for $(W-1)$ epochs in the preceding window of size W. Initialization based on these estimates will improve convergence. In addition, we will consider multi-state component models with multiple test outcomes.

Acknowledgements

The study reported in this paper was supported, in part, by the Toyota Technical Center and the Office of Naval Research under contract no. 00014-00-1-0101. Any opinions expressed in this paper are solely those of the authors and do not represent those of the sponsor.

References

[1] F. Yu, F. Tu, H. Tu, and K. R. Pattipati, "Multiple disease (fault) diagnosis with applications to the QMR-DT problem," *Proceedings of Computing Communication and Control Technology International Conference*, Austin, TX, 2004. To appear in *IEEE Trans. on SMC: Part A*, September 2007.

[2] N. Odintsova, I. Rish, S. Ma. "Multifault Diagnosis in Dynamic Systems," *Proceedings of IM*, 2005.

[3] M. Shakeri, K. R. Pattipati, V. Raghavan and A. Patterson-Hine, "Optimal and near-optimal algorithms for multiple fault diagnosis with unreliable tests," *IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews*, vol. 28, no. 3, August 1998.

[4] T. Le and C. N. Hadjicostis, "Graphical inference methods for fault diagnosis based on information from unreliable sensors," *Proceedings of Intl. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1012-1017, Singapore, December 2006.

[5] J. Ying, T. Kirubarajan, and K. R. Pattipati, "A Hidden Markov model based algorithm for fault diagnosis with partial and imperfect tests," *IEEE Transactions on SMC: Part C - Applications and reviews*, vol.30, no. 4, 2000.

[6] O. Erdinc, C. Raghavendra, and P. Willett, "Real-time diagnosis with sensors of uncertain quality," *SPIE Conference Proceedings*, Orlando, pp. 1490– 1499, April 2003.

[7] S. Ruan , Y. Zhou , F. Yu , K. R. Pattipati, P. Willett and A. Patterson-Hine, "Dynamic multiple fault diagnosis and imperfect tests," *IEEE Transactions on Systems, Man and Cybernetics: Part A*, under review, June 2006.

[8] Z. Ghahramani and M. I. Jordan, Factorial hidden Markov models, Machine Learning, *Kluwer Academic Publishers*, Boston, 1997.

[9] S. Singh, S. Ruan, K. Choi, K. Pattipati, P. Willett, S. M. Namburu, S. Chigusa, D. V. Prokhorov and L. Qiao, "An optimization-based method for dynamic multiple fault diagnosis problem," *IEEE Aerospace Conference*, Big Sky, Montana, March 2007.

[10] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference, *Morgan Kauffmann Publishers Inc.*, San Mateo, CA, 1988.

[11] F. Tu, K. R. Pattipati, S. Deb, and V. N. Malepati, "Computationally efficient algorithms for multiple fault diagnosis in large graph-based systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 33, no. 1, pp. 73-85, January 2003.

[12] K. R. Pattipati and M. G. Alexandridis, "A heuristic search and information theory approach to sequential fault diagnosis," *IEEE Trans. on Systems Man and Cybernetics*, vol. 20, no. 4, pp. 872-887, 1990.

[13] V. Raghavan, M. Shakeri and K.R. Pattipati, "Optimal and near-optimal test sequencing algorithms with realistic test models," *IEEE Transactions on Systems, Man, and Cybernetics: Part A - Systems and Humans*, vol. 29, no. 1, January 1999, pp. 11-27.

[14] M. Shakeri, V. Raghavan, K. R. Pattipati, and A. Patterson-Hine "Sequential testing algorithms for multiple fault diagnosis," *IEEE Trans. on Systems, Man and Cybernetics: Part A*, vol. 30, pp. 1-14, Jan. 2000.

[15] J. E. Beasley, "An algorithm for set covering problems," *European Journal of Operational Research*, vol. 31, pp. 85-93, 1987.

[16] S. Singh, K. Choi, A. Kodali, K. Pattipati, S. M. Namburu, S. Chigusa, D. V. Prokhorov, and L. Qiao, "Dynamic fusion of classifiers for fault diagnosis," *IEEE SMC Conference*, Montreal, Canada, October 2007.

[17] D. Bertsekas, Nonlinear programming, *Athena Scientific*, Belmont, MA, USA, 2nd Edition, 2003.

Intermittent fault diagnosis: a diagnoser derived from the normal behavior

Siegfried Soldani* and **Michel Combacau** and **Audine Subias** and **Jérôme Thomas***
ssoldani@laas.fr combacau@laas.fr subias@laas.fr jerome.thomas@actia.fr
LAAS-CNRS, 7 Avenue du Colonel ROCHE FRANCE-31077 Toulouse Cedex 4
Université de Toulouse
*ACTIA, 25 Chemin de Pouvourvillle - B.P. 4215 FRANCE-31432 Toulouse Cedex 4

Abstract

This paper deals with an approach for the localization of intermittent faults in discrete events systems with partial observability. The proposed methods are based on a discrete events model representing the normal functioning of the observable behavior of the monitored system. This model based on automata formalism is built from the design data. The detection step consists of a comparison between the flow of observable events emitted by the monitored system and the flow foreseen by the model. A localization mechanism, based on the diagnoser approach, points out the set of events potentially responsible for the faults. These two mechanisms are designed in order to operate on board, in real time. An example from the automotive domain is presented.

Introduction

Nowadays, communication has been becoming a main element in the different electronic and computerized systems. The data exchanges play a fundamental part in the correct operation of the different devices of a control system. For this reason, the problem of fault detection and diagnosis at discrete events level is an important challenge in dealing with system failures.

Moreover, in many systems, faulty behavior often occurs intermittently. To detect this kind of fault, we have to consider an approach which will be developped to use on-line. Therefore, our proposal is an approach for the design of an on-board detection and diagnosis system suitable for any networked architecture and mostly for the intermittent faults.

This approach can be applied to different application fields such as in all transportation systems and particularly in the automotive field where a typical on-board control system is composed of different devices (sensors, actuators, processors) which exchange data through a communication bus to fulfill the functions required by the optimal operation of the vehicle.

Section two recalls the main work in the domain of discrete events diagnosis and positions our proposal. The main results about localization mechanisms are presented in section three. An application example is given in section four and the conclusion gives the most promising perspectives of this work.

Diagnosis and discrete events systems

Fault detection and diagnosis of discrete event systems have been the subject of many studies. Thus, some Petri nets approaches to fault diagnosis (Boubour *et al.* 1997; Genc & Lafortune 2003; Hadjicostis & Verghese 1999; Jiroveanu & Boel 2005; Lefebvre & Delherm 2005; Valette 1995), alarm correlation approaches (Jakobson & Weissman 1993; Nygate 1995) and diagnoser approaches (Contant, Lafortune, & Teneketzis 2002; Lamperti & Zanella 2003; Pencolé 2004; Sampath, Lafortune, & Teneketzis 1998),... have been proposed. In these research projects, the model of the system to diagnose is a behavioral model including both normal operation and faults. These approaches give very good results for predictable faults. Indeed, to be detected and diagnosed, a fault must be taken into account by the model of the system. It implies an accurate knowledge of the system and of the faults.

But, it is not always realistic to consider that all the faults can be known. The ongoing increase of electronic and computerized devices in the systems involves a difficulty in obtaining information about the faults which can occur on the system. So, it is impossible to foresee all the faults exhaustively. Therefore, we consider in our approach only the normal behavior of the system. Nevertheless we assume that a device failure may involve the occurrence of a faulty event or the lack of a correct event. Thus, we do not have any information about the failure, but we know some classes of faults. These classes of faults represent the insertion of spurious events or the lack of events.

These faults may result from the failures of the devices such as bad electrical contacts (e.g. faulty relays), sticky components (e.g. stuck valves), unknown bugs etc.. These failures are some typical situations of intermittent faults. This problem of intermittent faults is starting to be studied in much research (Contant, Lafortune, & Teneketzis 2004; Correcher *et al.* 2003; Jiang & Kumar 2006) but still remains to be explored in more detail. In these studies, the faulty events are assumed to be followed by the corresponding reset event, -this differs from our approach.

Currently, we are considering the lack of only one event or the occurrence of only one spurious event, these two situations being the typical symptoms of an intermittent fault. We assume, by this, that an intermittent fault is detected before the appearance of a new one.

Description of Localization reasoning

Our proposition consists of three steps: the modeling of system behavior (off-line) and the detection method of the intermittent and fugitive faults (on-line), which are described in previous studies (Soldani *et al.* 2006), and a localization method (on-line), presented here.

Previous Work

These last works deal with the modeling of system behavior and the detection method of intermittent and fugitive faults. This model is built from design data where only the observable events (communication events) are represented. The detection mechanism consists of a comparison between the flow of observable events emitted by the monitored system and the flow foreseen by the model. Then, a first reasoning of localization is developed and consists in modifying the sequence of last observable events leading to an inconsistency by deleting one of these events or by inserting an event of the model within this sequence. The goal is to restore the consistency between the observations and the model state trajectories. This reasoning, for example the calculation of the sequence size, is executed on line. In order to decrease the complexity of the on-line computation, we decided to develop an approach based on the diagnoser approach (Sampath, Lafortune, & Teneketzis 1998). Let us recall that the main difference between the classic diagnoser and our approach is that we do not know the exact faults but only some classes of faults. The difficulty is to determine all insertions and lacks, the consequences in the model behavior, and to build such a model. Our diagnoser takes all insertions and lacks in the normal behavior model into account.

Definitions and recalls

Now, let us recall some definitions on the automata and the synchronised product. We defined an automaton as :

Definition 1 ***labeled transitions system***

A labeled transitions system is a four-tuple $\Gamma =< S, S_0, E, R >$ *where:*

- *S is the finite set of states;*
- S_0 *is the finite set of initial states* $S_0 \subseteq S$*;*
- *E is the finite set of labels i.e. transition labels;*
- *R is the set of labeled transitions* $R \subseteq S \times E \times S$*, noted (s,a,s') or* $s \xrightarrow{a} s'$.

Remarks: We define ϵ the empty label as (s,ϵ,s).

Definition 2 ***Cartesian product of two automata***
Cartesian product of two automata $\Gamma_i =< S_i, S_{0i}, E_i, R_i >$, $i \in \{1, 2\}$ *is the automaton* $\Gamma =< S, S_0, E, R >$ *such as:*

- $S = S_1 \times S_2$*;*
- $S_0 = S_{01} \times S_{02}$*;*
- $E = E_1 \times E_2$*;*
- *R is the set of the transitions defined by* $((s_1, s_2), a, (s'_1, s'_2))$ *iff :*
  - *either* $a = (a_1, \epsilon)$ *and* $(s_1, a, s'_1) \in R_1$ *and* $(s_2, \epsilon, s'_2) \in R_2$*;*
  - *either* $a = (\epsilon, a_2)$ *and* $(s_1, \epsilon, s'_1) \in R_1$ *and* $(s_2, a, s'_2) \in R_2$*;*
  - *either* $a = (a_1, a_2)$ *and* $(s_1, a_1, s'_1) \in R_1$ *and* $(s_2, a_2, s'_2) \in R_2$*;*

Definition 3 ***Synchronised product of two automata***
The synchronised product of two automata Γ_1 *and* Γ_2*, noted* $\Gamma_1 \| \Gamma_2 \backslash Sync$ *is the Cartesian product restricted to the labeled transitions in Sync* $\subseteq (E_1 \times E_2)$*. This automaton is noted* $\Gamma =< S, S_0, Sync, R >$.

Fault Class Model

A fault class model is derived from the normal behavior model. As we have said previously, the building of the normal behavior function is not described here. This model is represented by an automaton and is noted by Γ_{BF}. We use in the next a simple example (figure 1) in order to present our approach.

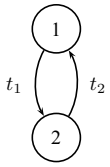


Figure 1: Γ_{BF} automaton

The fault class models do not take the faulty events into account but the set of all possible insertions and also all possible event lacking. In fact, we build two fault class models Γ^+ and Γ^- which represent respectively the assumption that an event may insert or be lacking. The automaton $\Gamma^+ =< S^+, S_0^+, E^+, R^+ >$ (cf. Figure 2) is defined by :

- $S^+ = \{OK, F^+_{t_1}, \ldots, F^+_{t_n}\}$,$\forall t_i \in E(\Gamma_{BF})$;
- $S_0^+ = \{OK\}$;
- $E^+ = \{t_i^+, \ldots, t_n^+\}$, $\forall t_i \in E(\Gamma_{BF})$;
- $R^+ = \{(OK, t_i^+, F^+_{t_i})\}$.

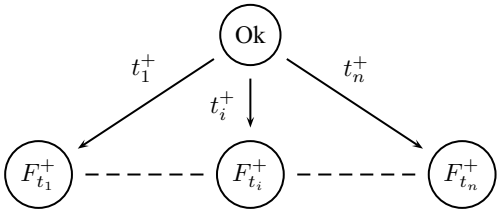


Figure 2: Insertion fault class model

To get the lacking fault class model, it is sufficient to change t_i^+ into t_i^- and $F^+_{t_i}$ into $F^-_{t_i}$. The insertion of an event e_i associated to the transition t_i is represented in Γ^+ by a transition noted by t_i^+. In the same way, the lack of an event e_i associated with the transition t_i is represented in Γ^- by a transition noted by t_i^-. Thus, $F^+_{t_i}$ is the assumption that the

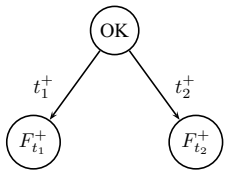


Figure 3: Γ^+ automaton

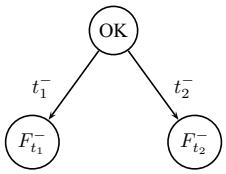


Figure 4: Γ^- automaton

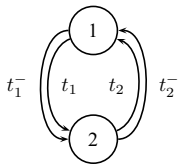


Figure 5: Γ_{BF}^- automaton

event e_i associated with the transition t_i is inserted. $F_{t_i}^-$ is the assumption that the event e_i associated to the transition t_i is lacking.
Remark1: n corresponds to the transitions number in the normal behavior model.
Remark2: we consider that only one event can insert or be lacking, which is represented in the corresponding models by the fact that no transition follows the states $F_{t_i}^+$ or $F_{t_i}^-$. Nevertheless, to consider the many possibilities, the models can be modified by adding transitions after these states.

For our example, we get two fault class models which are represented in figure 3 and 4. Figure 3 represents the fault class model Γ^+ for the insertion of events (accounted for the transitions t_i^+). In the same way, figure 4 represents the fault class model Γ^- for the lacking events. As we can notice, we take all transitions in Γ_{BF} model into consideration and only one insertion at a time is considered.

Model including generic faults

To get a model representing the evolution of these assumptions (lack and insertion), we modify the normal behavior model by considering these insertions or lacks.

Lacking event The lack of an event is considered as the occurrence of an unobservable event and therefore the system may evolve whereas the model remains in the same state. Thus, the modification for the lacking events consists in, for each transition in R_{BF}, defined by (s, t_i, s'), creating a new transition defined by (s, t_i^-, s'). The unobservable event is associated to the transition t_i^-. The new automaton, noted $\Gamma_{BF}^- =< S_{BF}^-, S_{0_{BF}}^-, E_{BF}^-, R_{BF}^- >$, is defined by :

- $S_{BF}^- = S_{BF}$;
- $S_{0_{BF}}^- = S_{0_{BF}}$;
- $E_{BF}^- = E_{BF} \bigcup \{t_i^-\}, \forall t_i \in E(\Gamma_{BF})$;
- $R_{BF}^- = R_{BF} \bigcup \{(s, t_i^-, s')\}$ such as :
  $\forall (s, t_i, s') \in R_{BF}, (s, t_i^-, s') \in R_{BF}^-$.

In our example, the model Γ_{BF}^- is represented in figure 5. Only the output transitions are represented in the sense where only the expected events can be lacking.

Insertion of event A received event may be faulty. If this event corresponds to an expected event then the model may evolve, whereas the system remains in the same state. Therefore, the modification for the inserted events consists in adding a transition in the automaton whose input state s is the same as the output state and whose label is t_i^+. It is noted (s, t_i^+, s). The faulty event must not change the state in the model. It is the same case as when we have the empty label. Note that we add to a state s only its output transitions because the events associated with these transitions are the only events that may change the trajectory without being immediately detected. That is why for each event occurrence, an insertion assumption about it is always made and not represented. The new automaton, noted by $\Gamma_{BF}^+ =< S_{BF}^+, S_{0_{BF}}^+, E_{BF}^+, R_{BF}^+ >$, is defined by :

- $S_{BF}^+ = S_{BF}$;
- $S_{0_{BF}}^+ = S_{0_{BF}}$;
- $E_{BF}^+ = E_{BF} \bigcup \{t_i^+\}, \forall t_i \in E(\Gamma_{BF})$;
- $R_{BF}^+ = R_{BF} \bigcup \{(s, t_i^+, s)\}$ such as :
  $\forall (s, t_i, s') \in R_{BF}, (s, t_i^+, s) \in R_{BF}^+$.

From the Γ_{BF} model, we build the Γ_{BF}^+ model (figure 6). We add some transitions to take the possibilty that an event can be spurious into account. As we can notice, we add only the output transitions of a state. Indeed, if the spurious event does not correspond to these transitions, it will be immediately detected since it is inconsistent with the expected events from this state. But, even if the other events are not represented in the model Γ_{BF}^+, that does not mean that we do not consider them. Each time that an event occurs, we assume that this event can be spurious. If there is no detection, this assumption is cancelled. Thus, it enables us to avoid representing all transitions in all states of our model Γ_{BF}.

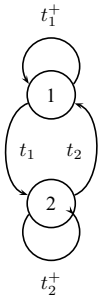


Figure 6: Γ_{BF}^+ automaton

Synchronised Product

After building these different models, a synchronised product is made between the fault class models and the modified normal behavior model. This synchronisation associates the function states (S_1,...,S_n) to modes (OK, $F_{t_i}^+$) or (OK, $F_{t_i}^-$)

where the function is or could be, depending on the transitions allowed in the product. For the lacking and inserted events, the synchronised products are done by:

$\Gamma^-_{Sync} = \Gamma^-_{BF} \| \Gamma^- \backslash Sync$ with Sync = $\left\{(t_i^-, t_i^-), (t_i, \epsilon)\right\}$
$\Gamma^+_{Sync} = \Gamma^+_{BF} \| \Gamma^+ \backslash Sync$ with Sync = $\left\{(t_i^+, t_i^+), (t_i, \epsilon)\right\}$

We get all parameters defined by the synchronised product, noted $S^{(+/-)}_{sync}, S^{(+/-)}_{0sync}, E^{(+/-)}_{sync}, R^{(+/-)}_{sync}$.
This is the point where our approach causes some troubles because the product generates a lot of states. We get for the class fault models $(n+1)$ states and n transitions. The synchronised product generates $n_s^+ \times (n+1)$ states and at most $(n_t^+ \times n)$ transitions (idem for the other product), where n_s^+ and n_t^+ is the number of states and transitions in Γ^+_{BF}. In fact, n_s^+ and n_s^- are equal to the number of states in normal model, i.e. equal to n, and in the same manner, n_t^+ and n_t^- are equal to 2n. Thus, we get a polynomial complexity for the computation ($n \times (n+1)$ states and $2n^2$ transitions). However, since the latter is made off line, it is possible to use this approach to localize the faulty event.
In our example, the synchronised product Γ^-_{Sync} (figure 7) is built from the model Γ^-_{BF} and Γ^- represented in figure 5 and in figure 4 respectively.
In the same way, the synchronised product Γ^+_{Sync} (figure 8) is built from the model Γ^+_{BF} and Γ^+ represented in figure 6 and in figure 3 respectively. This model represents

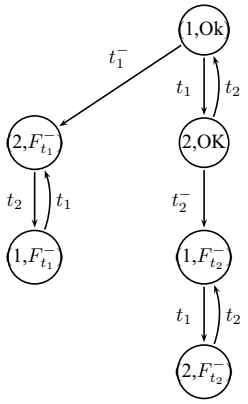


Figure 7: Γ^-_{Sync} automaton

the model evolution under different assumptions. We notice that the numbers of states and transitions are respectively $n_s^+ \times (n+1) = 2 \times 3 = 6$ states and $n_t^+ \times n = 4 \times 2 = 8$ transitions for the Γ^+_{Sync} model. The numbers of states and transitions are respectively $n_s^- \times (n+1) = 2 \times 3 = 6$ states and $n_t^- \times n = 4 \times 2 = 8$ transitions for the Γ^-_{Sync} model.

Diagnoser

Nevertheless, we need to build two new models, Γ^+_{Diag} and Γ^-_{Diag} from Γ^+_{Sync} and Γ^-_{Sync} which take the meaning of the

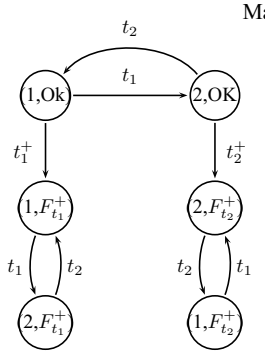


Figure 8: Γ^+_{Sync} automaton

transitions t_i^+ and t_i^- into account. Moreover, this model also takes into account the fact that some states will never be reached because they do not correspond to the normal behavior and a detection will be made before. We obtain two diagnosers, one for the lacking events, the other for the inserted events.

Insertion of events A transition t_i^+ is associated with the same event as the one associated with the transition t_i. Thus, from a state in the synchronised model, we have to gather the output states of transitions t_i and t_i^+ because they correspond to the same event (which is seen on the communication bus).

So we can define different rules to build the new model from the previous synchronised model so as to take this condition into account. The new model $\Gamma^+_{Diag} =< S^+_{Diag}, S^+_{0_{Diag}}, E^+_{Diag}, R^+_{Diag} >$ is caracterized by:

- $S^+_{Diag} \subseteq \mathcal{P}(S^+_{Sync})$, set of subsets of S^+_{Sync};
- $S^+_{0_{Diag}} = \left\{S^+_{0_{Sync}}\right\}$;
- $E^+_{Diag} \subseteq E^+_{Sync}$;

First of all, we can initialize the set S^+_{Diag} and R^+_{Diag} by:

- $S^+_{Diag} = \{\{s_i\}\}, \forall s_i \in S^+_{Sync} = S^+_{BF} \times S^+$;
- $R^+_{Diag} = \{(\{s_i\}, t, \{s_j\})\}, \forall (s_i, t, s_j) \in R^+_{Sync}$.

Now, we can define these different rules to build our diagnoser:

1. Let x, x' $\in R^+_{Diag} | x = (\{s_i\}, t_i, \{s_j\})$ and $x' = (\{s_i\}, t_k, \{s'_j\})$,
   $t_k = t_i^+ \Rightarrow$
   - $S^+_{Diag} = S^+_{Diag} \cup \left\{\left\{s_j, s'_j\right\}\right\}$
   - $R^+_{Diag} = R^+_{Diag} \cup \left\{(\{s_i\}, t_i, \left\{s_j, s'_j\right\})\right\}$
   - $R^+_{Diag} = R^+_{Diag} \backslash \{x, x'\}$
2. Let $s_m = (., OK) \in S^+_{Sync}$ and $S_1, S_2 \in S^+_{Diag}$
   $s_m \in (S_1 \cap S_2) \Rightarrow$
   - $S^+_{Diag} = S^+_{Diag} \cup \{S_1 \cup S_2\} \backslash \{S_1, S_2\}$

- And $\forall S_x \in \{S_1, S_2\}$,
  - $(S_x, t, S_y) \in R^+_{Diag} \Rightarrow R^+_{Diag} = R^+_{Diag} \cup \{(\{S_1 \cup S_2\}, t, S_y)\} \setminus \{(S_x, t, S_y)\}$
  - $(S_y, t, S_x) \in R^+_{Diag} \Rightarrow R^+_{Diag} = R^+_{Diag} \cup \{(S_y, t, \{S_1 \cup S_2\})\} \setminus \{(S_y, t, S_x)\}$

3. Let $S_i, S_j, S_m, S_l \in S^+_{Diag}$ and x,x' $\in R^+_{Diag} | x = (S_i, t_k, S_l), x' = (S_j, t_l, S_m)$,
   - $S_j \subseteq S_i \Rightarrow R^+_{Diag} = R^+_{Diag} \cup \{(S_i, t_l, S_m)\}$
   - $S_j \subseteq S_i$ and $t_k = t_l \Rightarrow$
     - $R^+_{Diag} = R^+_{Diag} \cup \{(S_i, t_k, \{S_l \cup S_m\})\}$
     - $S^+_{Diag} = S^+_{Diag} \cup \{S_l \cup S_m\}$
     - And $\forall S_x \in \{S_l, S_m\}$,
       - $(S_x, t, S_y) \in R^+_{Diag} \Rightarrow R^+_{Diag} = R^+_{Diag} \cup \{(\{S_l \cup S_m\}, t, S_y)\} \setminus \{(S_x, t, S_y)\}$
       - $(S_y, t, S_x) \in R^+_{Diag} \Rightarrow R^+_{Diag} = R^+_{Diag} \cup \{(S_y, t, \{S_l \cup S_m\})\} \setminus \{(S_y, t, S_x)\}$

4. Let $x = (S_i, t, S_j) \in R^+_{Diag}$ $(., ok) \notin S_i \Rightarrow R^+_{Diag} = R^+_{Diag} \setminus \{x\}$

The last rule does not allow one to build the following of any set of states in which the state "(.,ok)" is not included. Indeed, that means that the model is no longer in the normal behavior and the detection of a problem is made before. However, the rule does not say that we have only one fault assumption. We can have several fault assumptions in the terminal set. In fact, we do not try to distinguish the different fault assumptions and therefore to be diagnosable. If we want to do a study of diagnosability, this rule must be removed or modified to stop the development of the graph as soon as we have only one fault assumption in our set of states. This set has then a single state which correspond to the fault assumption (.,F^+_{ti}).
The diagnoser model Γ^+_{Diag} is built by applying these differents rules to our synchronised model Γ^+_{Sync}. For our example, this insertion diagnoser is represented in figure 9. We can notice that each time that an event occurs (associated with t_1 or t_2), this implies an insertion assumption which is grouped with the assumption of normal behavior. Moreover, each time we have a state with only a fault assumption, we do not develop a single set of states.

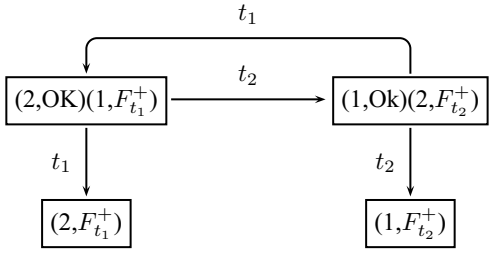


Figure 9: Γ^+_{Diag} automaton : Insertion Diagnoser

Lacking events An event/transition t^-_i corresponds to a lacking event and therefore is an unobservable event/transition. Thus, from an observation point of view, if $\exists(s_j, t^-_i, s'_j)$, s_j and s'_j cannot be distinguished and are grouped together in Γ^-_{Diag}. The model Γ^-_{Diag} is characterized and initialized in the same way as for Γ^+_{Diag} but from the Γ^-_{Sync} elements. The set of rules used to build Γ^-_{Sync} is constituted by the rule 1 defined below and rules 2,...,4 defined above.

1. Let $S_y, S_x, S_i, S_j \in S^-_{Diag}$ with $S_x \in \{S_i, S_j\}$, $S_y \notin \{S_i, S_j\}$
   - $x = (S_i, t^-_n, S_j) \in R^-_{Diag} \Rightarrow S^-_{Diag} = S^-_{Diag} \cup \{S_i \cup S_j\}$ and $R^-_{Diag} = R^-_{Diag} \setminus \{x\}$
   - $x = (S_x, t, S_y) \in R^-_{Diag} \Rightarrow R^-_{Diag} = R^-_{Diag} \cup (\{S_i \cup S_j\}, t, S_y) \setminus \{x\}$
   - $x = (S_y, t, S_x) \in R^-_{Diag} \Rightarrow R^-_{Diag} = R^-_{Diag} \cup (S_y, t, \{S_i \cup S_j\}) \setminus \{x\}$

The diagnoser model Γ^-_{Diag} is built by applying these differents rules to our synchronised model Γ^-_{Sync}. This lacking diagnoser of the example is represented in figure 10. We can notice that each time that an event occurs (associated to t_1 or t_2), this implies a lack assumption which is grouped with the assumption of normal behavior. A lacking event is seen as an unobservable event and therefore we can not distinguish the state linked by these events. Moreover, each time we have a state with only a fault assumption, we do not develop a single set of states.

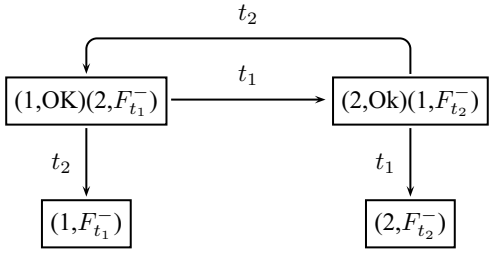


Figure 10: Γ^-_{Diag} automaton : Lacking Diagnoser

Application in automotive industry

Specificities of the automotive context

Over the last years, the number of security systems, aiding systems, options for the drivers comfort etc. have drastically increased. This evolution is due to the use of electronic devices connected by one or several local area networks. In most architectures, the devices called Electronic Control Units (ECUs) are connected by a specific local network: the Controller Area Network (CAN). CAN is based

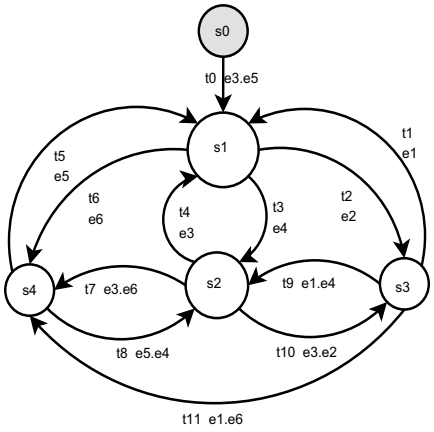


Figure 11: "front wiping" abstracted model

on a serial communication protocol, which supports distributed real-time control. The functions provided to the passengers are implemented by a collaboration of the different ECU. Our example concerns the "front wiping" function. The model of this function is not of a high level of complexity, but it is sufficient to show how our proposal works.

This function is distributed on three different ECU: the "on/off and speed selection" ECU, the actuators ECU and the main ECU. The latter manages the behavior of the function. It receives messages from the two other ECUs and sends control messages to the actuators. These ECUs are designed to deal with some foreseen faults and not for intermittent faults. Our work is intended to give a solution to this kind of faults. In this architecture, the different ECUs are "black boxes" and the real time monitoring can only be done through the observation of the messages exchanged on the network. Let us see in this example how the localization mechanisms previously described can give a great advantage.

Modeling of a distributed function

The Γ_{BF} model (figure 11) represents the exchange of messages between the three ECU during normal operation of the front-wiping function and is built from design data. This function has five states:

s_0 : initial state reached each time the power is switched off,

s_1 : stand-by (wiping suspended or stopped),

s_2 : low-speed wiping,

s_3 : maintenance,

s_4 : high speed wiping.

The transitions are labeled by sets of events whose individual significations are the following:

e_1 : maintenance request off,

e_2 : maintenance request on,

e_3 : low speed request off,

e_4 : low speed request on,

e_5 : high speed request off,

e_6 : high speed request on.

Detection of a fault

Let us consider the initial state s1, and the sequence of events $[e_2, e_3]$. The event e_2 is consistent with the events expected in the model, but the following event e_3 does not belong to the set of expected events. So, the detection is made. Instead of reasoning from this sequence in order to determine which might be the faulty event, we use a diagnoser which provides us the different assumptions, which are consistent with the received events sequence. Otherwise, only an insertion assumption could explain an inconsistency.

Fault localization model

Building Γ^+_{BF}, Γ^-_{BF}, Γ^+_{Sync} and Γ^-_{Sync} leads to Γ^+_{Diag} and Γ^-_{Diag}. In this paper, only Γ^-_{Diag} is represented (figure 12). We can see that, from the state $\{(s_1, OK), (s_2, F^-_{t3}), (s_4, F^-_{t6}), (s_3, F^-_{t1})\}$, the sequence of events $[e_2, e_3]$ is consistent with a trajectory by pointing out the assumption (s_1,$F^-_{t_9}$). This means that the event associated with the transition t_9 i.e. the event $e_1.e_4$ is lacking. This is a valid assumption to explain the observed sequence.
An other valid assumption, which is not presented in this section, would be to consider that the event e_3 is a spurious event.
With this method, we reach the same results (fault localization) as those presented in (Soldani *et al.* 2006). However, the on-line computation complexity that only requires to follow the evolution of two DES models has been significantly reduced.

Conclusion

The aim of this paper is to describe a method for localization of intermittent faults by an on-board monitoring system based on discrete events models.
The monitoring system operates with a model of normal behavior only. The approach considered is very interesting for automotive applications, a domain in which intermittent faults lead to very awkward situations for mechanics. The localization of the fault, i.e. the lacking event or the spurious event, gives some potentially useful information to mechanics (the components at the origin of the fault). Indeed, a bad electrical contact may be misread by the electronic control unit. Thus, the latter may send a spurious event which we localize. And therefore, we can determine the component which sends it.
Moreover, in futur work, we want to save the maximum of information about the context in order to retrieve the conditions of appearance of the fault.
Technically, this on-board monitoring system is designed to be connected to the off-board diagnosis tool used by the mechanics in the garage.
Futur work will focus on the use of the Petri net to reduce

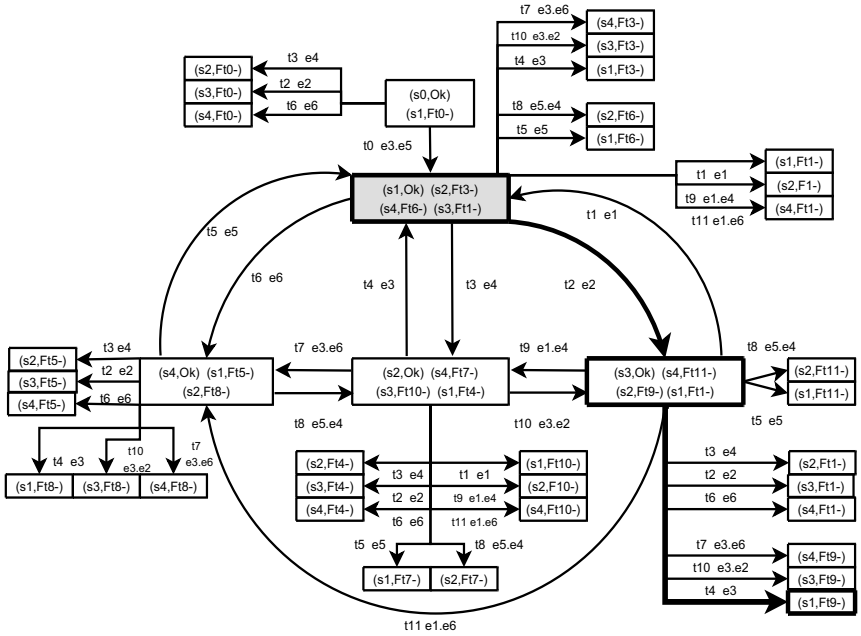


Figure 12: Lacking diagnoser model

the explosion of state in the synchronised product while remaining a model close to the diagnoser.
Moreover, we may adress the problem of model initialisation and reset when detection is made (Caines, Greiner, & Wang 1988; Giua 1997). Besides, with this approach, detectability and diagnosability analysis could be carried out, a similar approach to various works (Sampath, Lafortune, & Teneketzis 1998; Pencolé 2004).
A real application on a vehicle is being developed to prove the efficiency of the proposal.

Acknowledgments

The authors want to thank the French Ministry of Industry, the French Région Midi-Pyrénées and the FEDER program of the EEC for their support.

References

Boubour, R.; Jard, C.; Aghasaryan, A.; Fabre, E.; and Benveniste, A. 1997. A Petri net approach to fault detection and diagnosis in distributed systems. Part I: Application to telecommunication networks, motivations and modeling. In *Proceedings of the 36th IEEE Conference on Decision and Control*, 720–725.

Caines, P.; Greiner, R.; and Wang, S. 1988. Dynamical logic observers for finite automata. In *Proceedings of the 27th IEEE Conference on Decision and Control*, 226–233.

Contant, O.; Lafortune, S.; and Teneketzis, D. 2002. Failure diagnosis of discrete event systems: The case of intermittent faults. In *Proceedings of the 41st IEEE Conference on Decision and Control*, 4006–4011.

Contant, O.; Lafortune, S.; and Teneketzis, D. 2004. Diagnosis of intermittent faults. *Discrete Event Dynamic Systems* 14:171–202.

Correcher, A.; Garcia, E.; Morant, F.; Quiles, E.; and Blasco-Gimenez, R. 2003. Intermittent failure diagnosis in industrial processes. In *IEEE International Symposium on Industrial Electronics, ISIE'03*, volume 2, 4006–4011.

Genc, S., and Lafortune, S. 2003. Distributed diagnosis of discrete-event systems using Petri nets. In *ICATPN'03*, 316–336.

Giua, A. 1997. Petri net state estimators based on event observation. In *Proceedings of the 36th IEEE Conference on Decision and Control*, 4086–4091.

Hadjicostis, C., and Verghese, G. 1999. Monitoring discrete event systems using Petri net embeddings. In *ICATPN'99*, 188–207.

Jakobson, G., and Weissman, M. 1993. Alarm correlation. *IEEE Network* 7(6):52–59.

Jiang, S., and Kumar, R. 2006. Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications. 3:47–59.

Jiroveanu, G., and Boel, R. 2005. Petri net model-based distributed diagnosis for large interacting systems. In *Proceedings of the 16th International Workshop on Principles of Diagnosis, DX'05*.

Lamperti, G., and Zanella, M. 2003. Continuous diagnosis of discrete-event systems. In *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, 105–112.

Lefebvre, D., and Delherm, C. 2005. Diagnosis with causality relationships and directed paths in Petri net models. In *IFAC World Congress'05*.

Nygate, Y. A. 1995. Event correlation using rule and object based techniques. In *Proceedings of the 4th Symposium on Integrated Network Management*, 290–301.

Pencolé, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004*, 43–47.

Sampath, M.; Lafortune, S.; and Teneketzis, D. 1998. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control* 43(7):908–929.

Soldani, S.; Combacau, M.; Thomas, J.; and Subias, A. 2006. Intermittent fault detection through message exchanges: a coherence based approach. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'2006)*, 1549–1554.

Valette, R. 1995. Petri nets for control and monitoring : Specification, verification and implementation. In *Workshop on Analysis and Design of Event-Driven Operations in Process Systems, ADEDOPS*.

Diagnosing Dependent Failures – an Extension of Consistency-based Diagnosis *

Jörg Weber and **Franz Wotawa**
Technische Universität Graz
Institute for Software Technology
8010 Graz, Inffeldgasse 16b/2, Austria
Tel: +43 316 873 5723, Fax: +43 316 873 5706
{jweber,wotawa}@ist.tugraz.at

Abstract

Most applications of model-based diagnosis rely on the assumption that components fail independently. However, dependent failures are quite common in some domains. In such domains, the failure of one component may cause the failure of other components. Hence, multiple faults are much more likely, and the minimal diagnoses may not contain all components which have failed. In this work we propose the concept of *diagnosis environments (DEs)*. DEs distinguish between independent and dependent failures, and they may consider components as failed which are not part of the minimal diagnoses. We provide a formalization of our approach and present an algorithm which computes all *minimal DEs*.

Introduction

The applications of model-based diagnosis (MBD) usually rely on the assumption that components fail independently (Reiter 1987)(de Kleer & Williams 1987). However, as de Kleer remarks in (de Kleer 1990), dependent failures are quite common in some domains. In such domains the failure of one component may lead to the failure of other components as well. A component c_j fails in dependence of another component c_i if (1) c_i fails and (2) the failure of c_i causes some "damage" to c_j which prevents c_j from behaving as desired and (3) the damage to c_j is permanent, i.e., c_j exhibits abnormal behavior even after a repair of c_i, and so c_j must be repaired as well. If these 3 conditions hold, then we consider c_j as *abnormal*, even though its failure has been caused by external circumstances instead of an internal fault.

For example, from our experience with diagnosis in mobile autonomous robots we know that dependent failures often occur in hardware-software hybrid systems. There are different reasons for this. First, software components may be tightly coupled with hardware components, and so a failure of the latter may have a fatal impact on the former. A typical example is the failure of a camera which causes a crash of the image processing software. Another reason is that software components are often based on the same underlying framework (e.g., CORBA), and a failure in the framework may propagate to those components. A third reason is the communication over channels which do not strictly decouple the involved components. E.g., when a component attempts to make a remote procedure call to a failed component, then the former component may also fail.

A more detailed discussion of dependent failures in the hardware-software hybrid system of a mobile autonomous robot is provided in (Weber & Wotawa 2007). A component which has failed due to an internal error is an *independently failed component*, whereas a component whose failure has been caused by another component is a *dependently failed component*. A dependently failed component may cause the failure of further components; i.e., there can be a cascade of dependent failures.

Dependent failures may also occur in physical systems: suppose a component c_i fails and produces extra heat (or current, pressure, etc.) which causes physical damage to an adjacent component c_j.

Due to the assumption that components fail independently, most MBD systems compute only minimal diagnoses. However, in the next section we will give an example which shows that, in case of dependent failures, the minimal diagnoses often do not contain all components which have failed. Furthermore, the knowledge of the causal order of failures is often crucial for a successful repair of the system.

In order to tackle these problems, we propose the concept of diagnosis environments (DEs) and minimal diagnosis environments (MDEs). Each DE corresponds to a (in general not minimal) diagnosis; i.e., it may consider components as independently or dependently failed even if they are not abnormal in all minimal diagnoses. Moreover, a DE states the causal order in which the components have failed by distinguishing between independent and dependent failures.

MBD systems employ a system description and observations in order to compute the diagnoses. In addition, our approach relies on a *failure dependency graph* which indicates possible failure dependencies. This graph can be seen as part of the system model. As it contains all possible dependencies (i.e., it considers the worst case), we need to dynamically determine which of the possible dependent failures may actually have occurred. For this purpose, we introduce the *dependent failure description* which is a logical description of system behavior in case of dependent failures. This description is used to eliminate implausible DEs.

To the best of our knowledge, our proposal is the first work within the field of consistency-based diagnosis which addresses the issue of dependent failures at the level of log-

*This research has been funded in part by the Austrian Science Fund (FWF) under grant P17963-N04. Authors are listed in alphabetical order.

ical reasoning. In (Weber & Wotawa 2007) we introduced the idea of DEs and provided examples from the diagnosis in mobile autonomous robots. We also presented the results of case studies in which the independent failure of one component caused the failure of many other components. Our diagnosis system computed the MDEs and was able to successfully repair a large part of the system, whereas a repair based solely on minimal diagnoses would not have succeeded in this situation.

The main contribution of this paper is a formalization of our proposal which is application-independent and based upon Reiter's diagnosis framework (Reiter 1987). We also prove some important consequences of our definitions. Furthermore, we provide an algorithm which computes all MDEs of a system.

The following section provides an informal introduction to our proposal by means of an example. Then we formalize our approach and provide an algorithm which computes all MDEs for a system. Finally we provide a discussion and relate our research to the work of others.

Motivation

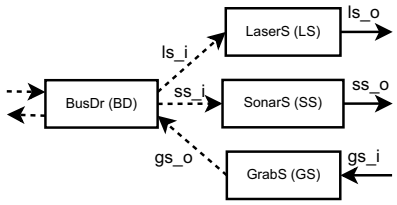


Figure 1: Software components of a robot control system.

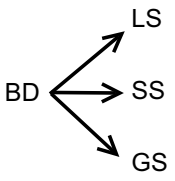


Figure 2: The failure dependency graph.

Figure 1 depicts several software components of the control system of a mobile autonomous robot. The directed connections indicate data flows between the components. Observable connections are depicted by solid lines, whereas dotted lines represent unobservable ones. The data on unobservable connections cannot be monitored by the diagnosis system.

BD is a service which controls the hardware bus and receives/transmits data from/over the bus. LS and SS rely on BD to receive data from sensors which are connected to the bus. These components process the raw sensor data and dispatch the results over their output connections, ls_o and ss_o. Finally, GS controls the grabber arm of the robot. The connections between BD and the three other components are not observable, because the latter perform remote calls to procedures of BD in order to receive/transmit data, and these calls cannot be intercepted.

As usual in MBD, we provide a logical system description (SD) which captures the nominal behavior of components. The predicate ab denotes *abnormal*:

$$\neg ab(LS) \rightarrow \neg term(LS)$$
$$\neg ab(LS) \wedge trans(ls_i) \rightarrow trans(ls_o)$$
$$\ldots$$

$\neg term(LS)$ holds iff c spawns all of its vital processes/threads ($term$ means "terminated"), and $trans(e)$ is true iff (arbitrary) data is transmitted over connection e. The models of SS and GS are analogous to LS.

Suppose that BD crashes. Then LS and SS will fail as well when they attempt to make a remote procedure call to the failed component BD: we assume that they throw software exceptions which are not handled, and so they terminate. Thus, the set of observations (OBS) contains the following literals: $\{term(LS),\ term(SS),\ \neg trans(ls_o),\ \neg trans(ss_o), ..\}$. There are no observations for the unobservable connections. The failure of BD is an independent failure, whereas LS and SS have failed dependently. However, in our scenario GS has not yet failed, as it makes only sporadic calls to BD.

We assume that the failure of BD cannot be directly determined due to the limited observability. Thus, we get the set of minimum diagnoses $\mathfrak{D} = \{\Delta\}$ with $\Delta = \{ab(LS),\ ab(SS)\}$[1]. Now we can try to repair the system. The authors of (Steinbauer & Wotawa 2005) propose to repair faults in a robot control system by restarting failed components. However, in our case the repair will fail if it is solely based on Δ: as BD is not repaired before, the two restarted components will immediately fail again when attempting to make a remote procedure call to BD.

It should be noted that the example system could also modeled in another way which considers the termination of LS and SS as a nominal reaction to a failure of BD, i.e., the termination of the former two components is not an abnormal behavior when it is caused by BD. In this case we would get 2 minimal diagnoses, namely $\{BD\}$ and $\{LS, SS\}$. Both are not satisfying, as they do not contain all components which are permanently "damaged" and which must be restarted.

The crucial point is that in systems with dependent failures the minimal diagnoses often do not contain all components which have failed. The focus on minimal diagnoses relies on the failure independency assumption. The usage of non-minimal diagnoses for repair is no solution for this problem, as their number may be very large, and they do not take failure dependencies into account. Furthermore, even the diagnosis $\{ab(BD), ab(LS), ab(SS)\}$ is not satisfying, because it does not indicate the failure dependencies which influence the order in which the components must be repaired (BD must be repaired before LS and SS), and because it misleadingly indicates that LS and SS have failed due to internal errors (software bugs in this example).

In order to tackle these issues, we propose the concept of *diagnosis environments (DEs)*. DEs are based on (not necessarily minimal) diagnoses; i.e., there is a set of DEs for each diagnosis, and the diagnoses are computed before creating their DEs (e.g., by Reiter's Hitting Set algorithm). As described below, DEs embody the causal order of failures. For this purpose we augment the system model with a *failure dependency graph (FDG)* whose directed edges represent possible failure propagations, see Fig. 2. An edge from component c_i to c_j states that a failure of c_i *may* directly cause a failure of c_j. Note that this does not mean that c_j *must* always fail when c_i fails.

For each component c, a DE comprises one of the literals $\neg ab(c)$, $indf(c)$, or $df(c_i, c)$, where $indf(c)$ indicates an in-

[1]For brevity we omit the $\neg ab(\cdot)$ literals in diagnoses and DEs.

dependent failure of c, whereas $df(c_i, c)$ states that c has dependently failed as a direct consequence of the failure of c_i. The literal $ab(c)$ does not occur in DEs, because any component which is abnormal has failed either independently or dependently. In other words, we regard the abnormality of a component as the consequence of its independent or dependent failure. For all components c which are abnormal in a diagnosis, either $indf(c)$ or $df(\cdot, c)$ is in all DEs of this diagnosis. Moreover, taking the failure dependency graph into account, a DE may consider components as failed which are not abnormal in the diagnosis the DE refers to.

In this example, there are 7 possible DEs of $\Delta = \{ab(LS), ab(SS)\}$[1]: $\theta_1 = \{indf(LS), indf(SS)\}$, $\theta_2 = \{indf(BD), df(BD, LS), indf(SS)\}$, $\theta_3 = \{indf(BD), df(BD, SS), indf(LS)\}$, $\theta_4 = \{indf(BD), df(BD, LS), df(BD, SS)\}$. The DEs θ_5, θ_6, θ_7 are equal to θ_2, θ_3, θ_4, respectively, with the only difference that they also contain $df(BD, GS)$, because GS might also have failed due to the failure of BD.

As multiple independent failures are unlikely in most applications, we focus on *minimal DEs (MDEs)* which have a minimal number of $indf(\cdot)$ literals. Thus, only θ_4 and θ_7 are minimal. However, the number of MDEs can still be large, and so we should seek to logically eliminate MDEs which are implausible.

We can achieve this by specifying abnormal behavior as well (Struss & Dressler 1989). For example, suppose only LS fails (i.e., it fails independently), and $OBS = \{\neg trans(ls_o), trans(ss_o), ..\}$. Then the set of minimal diagnoses contains, among others, $\Delta = \{ab(LS)\}$. So $\theta = \{indf(BD), df(BD, LS), ..\}$ may be a MDE of Δ. Hence, we improve our model and add

$$ab(BD) \rightarrow \neg trans(ls_i) \wedge \neg trans(ss_i)$$
$$trans(ls_o) \rightarrow trans(ls_i)$$
$$trans(ss_o) \rightarrow trans(ss_i)$$

to SD. This captures our experience that, when BD fails, usually no more data is transmitted over its outputs. Now, as $trans(ss_o)$ is observed, which implies $trans(ss_i)$, the assumption $ab(BD)$ is inconsistent with $SD \cup OBS$. Now we incorporate the following *Dependency Mode Axioms (DMA)* [2]:

$$indf(c) \rightarrow ab(c)$$
$$df(c_i, c_j) \rightarrow ab(c_i) \wedge ab(c_j)$$

These axioms express that the abnormality of a component is a consequence of its independent or dependent failure. Consequently, the assumption $indf(BD)$ is inconsistent with $SD \cup DMA \cup OBS$, and there remains only one MDE for Δ, namely $\{indf(LS)\}$.

As a second possibility to reduce the number of MDEs we introduce the *dependent failure description (DFD)*. It constrains the system behavior in case of dependent failures. We illustrate this by means of the first scenario with $\Delta = \{ab(LS), ab(SS)\}$. We got 2 MDEs: $\theta_4 = \{indf(BD), df(BD, LS), df(BD, SS)\}$ and $\theta_7 = \{indf(BD), df(BD, LS), df(BD, SS), df(BD, GS)\}$. As we assume that GS has not failed, our goal is to eliminate θ_7. We know that GS terminates when it fails due to BD. So we add

$$df(BD, GS) \rightarrow term(GS)$$

to DFD, and now θ_7 is inconsistent with $SD \cup DFD \cup DMA \cup OBS$, and so only one MDE remains, namely θ_4.

[2]Free variables in axioms are implicitely universally quantified.

Diagnosis Environments (DEs)

In (de Kleer, Mackworth, & Reiter 1992), a system is a tuple $(SD, COMP, OBS)$. We extend this definition:

Definition 1 (System) *A system is a tuple $(SD, COMP, FDG, DFD, OBS)$. SD, DFD, and OBS, are sets of first-order sentences. $COMP$ is a finite set of components. FDG is a directed acyclic graph (DAG) whose nodes are components. There is exactly one node for every $c \in COMP$.*

As usual in Reiter's framework, the system behavior is described in SD using the ab predicate. We do not restrict the usage of this predicate, i.e., it may also be used to specify abnormal behavior. Hence, the minimal diagnoses do not characterize all diagnoses (de Kleer, Mackworth, & Reiter 1992). We assume that SD does not take into account that components may fail dependently, and that neither the $indf$ nor the df predicate occur in $SD \cup OBS$.

All examples in the rest of the paper refer to the failure dependency graph which is depicted in Fig. 3. We use $parents(c)$ to denote the set of all components which have an edge to c (e.g., $parents(D) = \{B, C\}$). For simplicity, we assume that FDG is acyclic. Of course, in practice there may be components with mutual failure dependencies. However, this is beyond the scope of this paper.

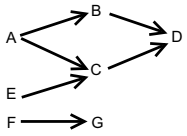


Figure 3: A failure dependency graph FDG.

As shown in the previous section, the dependent failure description DFD is used to eliminate implausible DEs. As the failure dependency graph models all possible failure dependencies, we use DFD to determine at runtime which of the possible dependent failures may actually have occurred. The intended usage of DFD is that it contains *dependent fault models*, i.e., sentences of the form $df(c_i, c_j) \rightarrow D_{ij}$, where D_{ij} is a description of the system behavior in case c_j fails in dependence of c_i.

If D_{ij} depends only on c_j for all $c_j \in COMP$, then it is possible to provide dependent failure descriptions for each component separately, i.e., these descriptions are device-independent. For example, D_{ij} may be a constraint on the input values of c_j, expressing that c_j may only fail dependently when receiving data whose value is out of a valid range. However, in many applications DFD will contain device-dependent descriptions.

The following definition states which modes a component may have in a DE. A component is either not abnormal or it has failed either independently or dependently:

Definition 2 ($dm(c)$ – Dependency Modes) *For all $c \in COMP$, $dm(c)$ denotes the set of literals $\{\neg ab(c), indf(c)\} \cup \{df(c_i, c) \mid c_i \in parents(c)\}$.*

Example 1 $dm(D) = \{\neg ab(D), indf(D), df(B, D), df(C, D)\}$, and $dm(F) = \{\neg ab(F), indf(F)\}$.

SD may contain descriptions of abnormal behavior, but DEs denote failed components by the $indf$ and df predicates. Hence, we define axioms which state that the abnormality of a component c is the consequence of its independent or dependent failure, and that a dependent failure of c can only be caused by an abnormal component c_i:

Definition 3 (DMA – Dependency Mode Axioms) *DMA is a set comprising the following two axioms:*

$$indf(c) \rightarrow ab(c)$$
$$df(c_i, c_j) \rightarrow ab(c_i) \wedge ab(c_j)$$

We regard diagnoses and DEs as *mode assignments*:

Definition 4 (Mode Assignment (MA)) *A mode assignment to a set of components $S \subseteq COMP$ is a set $\bigcup_{c \in S}\{m(c)\}$ with $m(c) \in M(c)$. $M(c)$ is a set of literals representing the possible modes of c. If $S = COMP$, the MA is* complete*, otherwise it is* partial*.*

$M(c)$ depends on the context. We introduce a shortcut: Let ω denote a MA to a set of components S. We define $\overline{\Gamma}(\omega) = \{c \mid \neg ab(c) \in \omega\}$, and $\Gamma(\omega) = S \setminus \overline{\Gamma}(\omega)$. In other words, $\Gamma(\omega)$ contains all components considered as failed in ω.

The following definition is analogous to Reiter's. The diagnoses are based only on $(SD, COMP, OBS)$:

Definition 5 (Diagnosis and Minimal Diagnosis) *A diagnosis Δ for $(SD, COMP, OBS)$ is a complete MA with $M(c) = \{\neg ab(c), ab(c)\}$ s.t. $SD \cup OBS \cup \Delta$ is consistent. Δ is* (subset-)minimal *iff there is no diagnosis Δ' s.t. $\Gamma(\Delta') \subset \Gamma(\Delta)$. Moreover, Δ is* empty *iff $\Gamma(\Delta) = \emptyset$.*

Chains are required for the subsequent definition of DEs:

Definition 6 (Chain) *Suppose ω is a MA with $M(c) = dm(c)$, $|\omega| = k$ with $k \geq 2$, and $\Gamma(\omega) = \{c_1, \ldots, c_k\}$ (i.e., $\overline{\Gamma}(\omega) = \emptyset$). Then ω is a* chain from c_1 to c_k *iff the following holds: $df(c_{i-1}, c_i) \in \omega$ for $i = 2, \ldots, k$.*

The notion of chain is similar to a path in graph theory. We use the notation $\{c_1 \rightsquigarrow c_k\} \subseteq \omega$ to denote that an improper subset of a MA ω is a chain from c_1 to c_k. Basically, $\{c_1 \rightsquigarrow c_k\} \subseteq \omega$ means that, according to ω, the failure of c_1 has directly or indirectly caused a failure of c_k: directly if $k = 2$, indirectly otherwise. Obviously, $\langle c_1, \ldots, c_k \rangle$ must be a path in FDG.

Example 2 Consider $\omega = \{indf(A),\ df(A, B),\ df(A, C),\ df(B, D)\}$. Several subsets of ω are chains: $\{indf(A),\ df(A, B),\ df(B, D)\}$, $\{indf(A),\ df(A, B)\}$, $\{df(A, B),\ df(B, D)\}$, $\{indf(A),\ df(A, C)\}$.

Definition 7 ($\Theta(\Delta)$ – Diagnosis Environments (DEs)) *For a diagnosis Δ, $\Theta(\Delta)$ denotes the set of all diagnosis environments of Δ. Each $\theta \in \Theta(\Delta)$ is a complete MA with $M(c) = dm(c)$ for all $c \in COMP$ s.t. the following conditions hold:*

1. *$\Upsilon \cup \theta$ is consistent, $\Upsilon := SD \cup DFD \cup DMA \cup OBS$*
2. *for all $c \in \Gamma(\Delta)$: $c \in \Gamma(\theta)$*
3. *for all c with $indf(c) \in \theta$: either $c \in \Gamma(\Delta)$ or there is a $c' \in \Gamma(\Delta)$ s.t. $\{c \rightsquigarrow c'\} \subseteq \theta$*

Υ is a useful shortcut which is used throughout the rest of this paper.

Note that this definition does not require Δ to be minimal. Moreover, it does not define minimal DEs (MDEs); MDEs will be defined below.

Item 2 of Def. 7 states that all components c which are abnormal in Δ are considered as failed in θ as well, i.e., either $indf(c) \in \theta$ or $df(\cdot, c) \in \theta$. Item 3 restricts which components may be assumed as independently failed: $indf(c)$ can only be in θ if either $ab(c) \in \Delta$ or c may have, directly or indirectly, caused a failure of any c' with $ab(c') \in \Delta$.

Def. 7 implicitely states which df modes may be in a DE:

Theorem 1 *If θ is a DE, then for a component c_l with $df(c_k, c_l) \in \theta$ there is exactly one component c with $indf(c) \in \theta$ and $\{c \rightsquigarrow c_l\} \subseteq \theta$.*

Proof. $DMA \cup \{df(c_k, c_l)\} \models ab(c_k)$. Hence, either $indf(c_k) \in \theta$ or there is a component c_j s.t. $df(c_j, c_k) \in \theta$. In the first case, $\{c_k \rightsquigarrow c_l\} \subseteq \theta$ holds, and we are done. In the second case, as above, we get that either $indf(c_j) \in \theta$ or there is a component c_i s.t. $df(c_i, c_j) \in \theta$, and $\{c_j \rightsquigarrow c_l\} \subseteq \theta$. This step can be repeated, but since the pathes in FDG are finite, the same also holds for chains, and so there must be at least one c with $indf(c) \in \theta$ s.t. $\{c \rightsquigarrow c_l\} \subseteq \theta$.
Moreover, as θ may contain at most one $df(\cdot, c')$ literal for each component c', there can be only one component c with $indf(c) \in \theta$ and $\{c \rightsquigarrow c_l\} \subseteq \theta$. □

Example 3 Suppose $\Delta = \{ab(B), ab(C)\}$ is a diagnosis. Then the following MAs are DEs of Δ, provided that they are consistent with Υ (Def. 7): $\{indf(B), indf(C)\}$, $\{indf(A), df(A, B), indf(C)\}$, $\{indf(A), df(A, B), df(A, C)\}$, $\{indf(A), df(A, B), df(B, D), indf(C)\}$, $\{indf(A), df(A, B), indf(E), df(E, C), df(C, D)\}$, etc. Note that neither F nor G are failed in any DE of Δ.

Example 4 Suppose $\Delta = \{ab(A), ab(F)\}$. Examples for DEs are: $\{indf(A),\ df(A, B),\ df(B, D),\ df(A, C),\ indf(F)\}$, $\{indf(A), df(A, C), df(C, D), indf(F)\}$, etc.

Example 5 Suppose $\Delta = \{ab(D)\}$. Then $\{indf(A),\ df(A, B),\ df(A, C),\ df(C, D)\}$ and $\{indf(E),\ df(E, C),\ df(C, D)\}$ are among the possible DEs. Again, F and G are not abnormal in all DEs of Δ.

We introduce another shortcut: for a MA ω, the σ function substitutes some modes in ω and returns the new MA. For example, $\sigma(\omega, ab/indf)$ substitutes every literal $ab(c)$ in ω by $indf(c)$, the other modes are not changed. Furthermore, $\sigma(\omega, ab(X)/indf(X))$ substitutes only the mode of a specific component X.

Intuitively, we expect to obtain a DE of a diagnosis Δ by substituting every literal $ab(c)$ in Δ by $indf(c)$. Our view is that, if a component behaves abnormally, then it is always possible that an internal error is the cause. In order to achieve this we impose formal restrictions on DFD:

Definition 8 (Restrictions on DFD) *The predicates* ab *and* indf *must not occur in DFD. Furthermore, DFD must not* interfere *with SD; i.e., it must be guaranteed that for any diagnosis Δ also $SD \cup DFD \cup OBS \cup \Delta$ is consistent.*

Since we also assume that the $indf$ predicate is not used in $SD \cup OBS$, this predicate only occurs in DMA. Now we can provide a theorem which complies with our intuitions:

Theorem 2 *If Δ is a diagnosis then $\theta = \sigma(\Delta, ab/indf)$ is a DE of Δ.*

Proof. Due to Def. 5 and Def. 8, $SD \cup DFD \cup OBS \cup \Delta$ is consistent. Because $\neg ab(c) \in \theta$ iff $\neg ab(c) \in \Delta$ and the $indf$ predicate does not occur in $SD \cup DFD \cup OBS$, we know that $SD \cup DFD \cup OBS \cup \theta$ is consistent. Finally, for all literals $ab(c)$ with $DMA \cup \theta \models ab(c)$ [i.e., $indf(c) \in \theta$], also $ab(c) \in \Delta$. Hence, $SD \cup DFD \cup DMA \cup OBS \cup \theta$ is consistent. The rest of the proof is trivial. □

Therefore, each diagnosis has at least one DE. The following two corollaries follow from Def. 7 and from Theorem 2:

Corollary 1 *Let Δ be a diagnosis. If FDG has no edges, then $\Theta(\Delta) = \{\theta\}$ with $\theta = \{\sigma(\Delta, ab/indf)\}$.*

Corollary 2 *If Δ is an empty diagnosis, then $\Theta(\Delta) = \{\Delta\}$.*

An important consequence of the definition of DEs is that each $\theta \in \Theta(\Delta)$ corresponds to a diagnosis Δ' with $\Gamma(\Delta') \supseteq \Gamma(\Delta)$:

Theorem 3 *If Δ is a diagnosis, then for all $\theta \in \Theta(\Delta)$ the following holds:*
$\Delta' = \left[\bigcup_{c \in \overline{\Gamma}(\theta)} \{\neg ab(c)\}\right] \cup \left[\bigcup_{c \in \Gamma(\theta)} \{ab(c)\}\right]$ *is also a diagnosis, and $\Gamma(\Delta') \supseteq \Gamma(\Delta)$.*

Proof. For all $c \in \Gamma(\theta)$: $DMA \cup \theta \models ab(c)$. Now it follows from Def. 7 that $\Upsilon \cup \theta \cup \{ab(c) \mid c \in \Gamma(\theta)\}$ is consistent, and so $SD \cup OBS \cup \Delta'$ is consistent, too. I.e., Δ' is a diagnosis. Moreover, from $c \in \Gamma(\Delta)$ it follows that that $c \in \Gamma(\theta)$, and $\Gamma(\theta) = \Gamma(\Delta')$. Hence, $\Gamma(\Delta') \supseteq \Gamma(\Delta)$. □

Now we extend the notion of DE to a set of diagnoses:

Definition 9 ($\Theta(\mathfrak{D})$ – DEs of a set of diagnoses) *If $\mathfrak{D}$ is a set of diagnoses, then $\Theta(\mathfrak{D}) = \bigcup_{\Delta \in \mathfrak{D}} \Theta(\Delta)$.*

It should be noted that for two different diagnoses Δ_1 and Δ_2 it is possible that $\Theta(\Delta_1) \cap \Theta(\Delta_2) \neq \emptyset$:

Example 6 Suppose $\Delta_1 = \{ab(A)\}$ and $\Delta_2 = \{ab(B)\}$. Then $\theta = \{indf(A), df(A, B)\}$ may be a DE of both Δ_1 and Δ_2, i.e., $\theta \in \Theta(\Delta_1)$ and $\theta \in \Theta(\Delta_2)$.

As the number of DEs can be daunting, we need a notion of *minimality*: minimal DEs have a minimal number of $indf$ modes, and they are, unlike DEs, only defined for minimal diagnoses. For the following definition we introduce a shortcut: for any MA ω, let $\#(\omega)$ denote the number of $indf(\cdot)$ literals in ω.

Definition 10 (Minimal DEs (MDEs)) *If $\mathfrak{D}$ is a set of* minimal *diagnoses, then $\theta \in \Theta(\mathfrak{D})$ is minimal in $\Theta(\mathfrak{D})$ iff the following holds for all $\theta' \in \Theta(\mathfrak{D})$: $\#(\theta') \geq \#(\theta)$.*

Example 7 Suppose $\mathfrak{D} = \{\Delta_1, \Delta_2\}$ with $\Delta_1 = \{ab(B), ab(C)\}$ and $\Delta_2 = \{ab(F)\}$, and $SD \cup OBS \cup \{ab(A)\}$ is inconsistent. Then $\Upsilon \cup \{indf(A)\}$ is inconsistent, too, and so $\#(\theta) = 2$ for all $\theta \in \Theta(\Delta_1)$, because all θ contain $indf(B)$ and either $indf(C)$ or $indf(E)$. Therefore, all DEs of Δ_1 are not minimal in $\Theta(\mathfrak{D})$, because $\{indf(F)\} \in \Theta(\mathfrak{D})$.

This example is very important, as it shows that there may be minimal diagnoses which do not contribute to the set of all MDEs of a system. Our algorithm which computes all MDEs for a system will benefit from this insight by computing only those diagnoses whose DEs may be minimal.

Clearly, if there is an empty diagnosis Δ (i.e., all components are not abnormal in Δ), then Δ is also a DE (see Corollary 2), and it is the sole MDE of the system. The following proposition states the minimum and maximum number of $indf$ modes for all DEs of a non-empty diagnosis:

Proposition 1 *For a non-empty diagnosis Δ the following holds for all $\theta \in \Theta(\Delta)$:*

$$1 \leq \#(\theta) \leq \#\big(\sigma(\Delta, ab/indf)\big) = |\Gamma(\Delta)|$$

Proof. From Def. 7 and Theorem 1 it follows that each DE of a non-empty diagnosis must contain at least one $indf(\cdot)$ literal. Furthermore, suppose $\#(\theta) > |\Gamma(\Delta)|$. Then, due to Item 3 of Def. 7, there must be components c, c' and c'' ($c' \neq c''$), s.t. the following holds: $c \in \Gamma(\Delta)$, $indf(c') \in \theta$, $indf(c'') \in \theta$, $\{c' \rightsquigarrow c\} \subseteq \theta$ and $\{c'' \rightsquigarrow c\} \subseteq \theta$. However, this conflicts with Theorem 1. □

Simple Algorithm

We first introduce a naive algorithm which computes all MDEs for a system, and describe important optimizations in the following section. We use $\mathfrak{D}$ to denote the set of all minimal diagnoses, and $\Theta(\mathfrak{D})$ is the set of all MDEs of the system.

The algorithm first computes the minimal diagnoses and then creates a *diagnosis environment graph (DEG)*. The DEG is a forest, i.e., it comprises disjoint trees, where each tree relates to a minimal diagnosis. The naive algorithm requires all minimal diagnoses, hence for $|\mathfrak{D}| = m$ the DEG contains m trees $T_1, .., T_m$. Each node n of one of the trees represents a MA, denoted by $n.\omega$, which may be a DE. More precisely, $n.\omega$ is a DE (but not necessarily a MDE) iff $\Upsilon \cup n.\omega$ is consistent.

Figures 4 and 5 depict the trees for the diagnoses $\Delta_x = \{ab(B), ab(C)\}$ and $\Delta_y = \{ab(A), ab(B)\}$, respectively. For any diagnosis Δ, the corresponding tree T_Δ has a root node n with $n.\omega = \sigma(\Delta, ab/indf)$. A node has the type α or β. The root node is an α-node. The children of an α-node may be of type α or β, whereas all children of a β-node are β-nodes as well. In Fig. 4 and 5, the edges to children of type α are depicted by solid lines, whereas the edges to β-children are dotted.

Algorithms 1 and 2 describe how a node is expanded. Remember that $parents(c)$ denotes the parents of c in FDG. Basically, when creating an α-child of n (i.e., a child of type α), a mode $indf(c) \in n.\omega$ is substituted by $df(c_i, c)$ and the mode of c_i may be set to $indf(c_i)$, whereas a β-child of n is generated by replacing a mode $\neg ab(c)$ by $df(\cdot, c)$.

It is important to note that these node expansion algorithms may create a new node n_j although the DEG already contains a node n_i with $n_i.\omega = n_j.\omega$. The nodes n_i and n_j may belong either to the same tree or to different trees. In both cases, n_j can be discarded (i.e., not added to the DEG), because the MAs of all descendant nodes of n_j (children, grandchildren, great-grandchildren etc.) are redundant. This is trivial to see if n_i and n_j have the same types, but it can be shown that this also holds if these nodes have different types (we omit the proof).

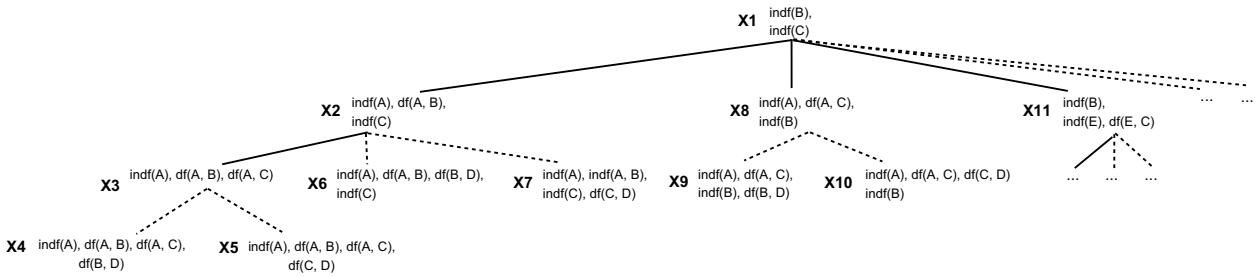


Figure 4: The DEG tree $T_{\Delta x}$ for $\Delta_x = \{ab(B), ab(C)\}$ containing only unique MAs. Nodes are labelled with X_i.

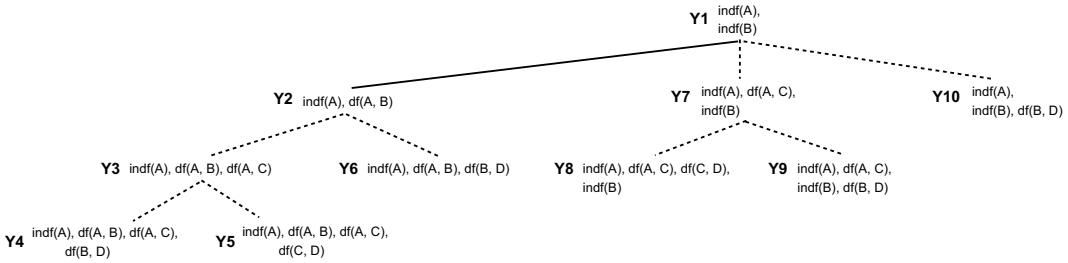


Figure 5: The DEG tree $T_{\Delta y}$ for $\Delta_y = \{ab(A), ab(B)\}$.

Algorithm 1: Create α-children of node n
Input: An α-node n
Output: A modified DEG containing new α-nodes
(1) For all c with $indf(c) \in n.\omega$:
(2) For all c_i with $c_i \in parents(c)$:
(3) create a new α-node n'
(4) $n'.\omega := \sigma(n.\omega, indf(c) / df(c_i, c))$
(5) If $\neg ab(c_i) \in n'.\omega$:
(6) $n'.\omega := \sigma(n'.\omega, \neg ab(c_i) / indf(c_i))$
(7) add n' to DEG (as child of n)

Algorithm 2: Create β-children of node n
Input: An α- or β-node n
Output: A modified DEG containing new β-nodes
(1) For all c with $\neg ab(c) \in n.\omega$:
(2) For all $c_i \in parents(c)$:
(3) If $c_i \in \Gamma(n.\omega)$:
(4) create a new β-node n'
(5) $n'.\omega := \sigma(n.\omega, \neg ab(c) / df(c_i, c))$
(6) add n' to DEG (as child of n)

Therefore, the algorithm should ensure that only unique MAs are generated. The simplest possibility is to compare each new MA with the already created nodes and to add the new node to the DEG only if its MA is unique.

Example 8 The MAs in Fig. 4 and 5 are unique *within* their trees. For example, the node X_8 could have an α-child with the MA $\{indf(A), df(A, B), df(A, C)\}$, but this MA would be equal to that of X_3. However, there are MAs which are depicted in both trees: Y_3 and its descendants are equivalent to X_3 and its descendants (and vice versa), and Y_7 and its descendants are equivalent to X_8 and its descendants (and vice versa). Thus, if $T_{\Delta y}$ is created after $T_{\Delta x}$, then the nodes Y_3 and Y_7 can be discarded, and $T_{\Delta y}$ contains only 4 nodes, namely Y_1, Y_2, Y_6, and Y_{10}.

A naive algorithm which computes all MDEs of a system works as follows. First, all minimal diagnoses of the system are generated, e.g. by using Reiter's Hitting Set algorithm. If there is an empty diagnosis, then this diagnosis is the sole MDE of the system (see Corollary 2), and the algorithm is finished. Otherwise, the entire DEG is created (with unique mode assignments). Afterwards, the algorithm performs consistency checks for all nodes n with $\#(n.\omega) = 1$, i.e., the consistency of $\Upsilon \cup n.\omega$ is checked. If consistent MAs are found, then they compose $\Theta(\mathfrak{D})$. Otherwise, the consistency of nodes with 2 $indf$ modes must be checked as well, etc. Note that it is guaranteed that the algorithm terminates, because the root nodes are DEs (see Theorem 2).

Improved Algorithm

A severe drawback of the naive algorithm is that it requires the generation of *all* minimal diagnoses and that it always creates the entire DEG. The improved algorithm seeks to cut the search space by creating only a part of the DEG. It initially assumes that there are MDEs with only one $indf$ mode (only if there is no empty diagnosis, of course), and so it does not expand a node n if it has more than one $indf$ modes and if it is clear that the same would hold for all descendants of n. In addition, if n is a root node, which directly stems from a minimal diagnosis Δ, then we know that all DEs of Δ contain more than one $indf$ mode, and so it is not even necessary to generate this diagnosis. In other words, when the diagnosis algorithm encounters the MA Δ,

then it does not need to check if Δ is a diagnosis. E.g., when Reiter's Hitting Set algorithm is used, the consistency check of $SD \cup OBS \cup \Delta$ and the expansion of the corresponding node in the HS-DAG can be omitted (Reiter 1987).

If there is no MDE with one $indf$ mode, then the algorithms searches for MDEs with two $indf$ modes, etc. So the generation of minimal diagnoses and the expansion of the DEG is an incremental process.

For this purpose, we introduce a function $\#_{\geq}(n)$ which returns a lower boundary for the number of $indf$ modes of n and of all of its descendants. Formally, this function returns a number ζ s.t. $\#(n.\omega) \geq \zeta$ and $\#(n'.\omega) \geq \zeta$ for all descendants n' of n. We use $\#_{\geq}(\Delta)$ as a shortcut for $\#_{\geq}(n)$ where n is a root node with $n.\omega = \sigma(\Delta, ab/indf)$.

Trivially, $\#_{\geq}(n)$ could always return 1, but we seek a better estimation. If n is a β-node, then $\#(n.\omega) = \#(n'.\omega)$ for all descendants n' of n. Thus, $\#_{\geq}(n) = \#(n.\omega)$. For α-nodes, an estimation can be obtained as follows: Suppose n has 2 $indf$ modes, namely $indf(c_i)$ and $indf(c_j)$. Then $\#_{\geq}(n) = 1$ if there is a directed path in FDG from c_i to c_j (or vice versa) or if there is a component c_k s.t. there are directed pathes in FDG from c_k to both c_i and c_j. This idea can be generalized to $\#(n.\omega) = k$ with $k \geq 2$. We do not show this here due to space reasons.

Example 9 Consider node X_1 in Fig. 4. As component A has pathes in FDG to both B and C, there may be descendants of X_1 with only one $indf$ mode, namely $indf(A)$, and so $\#_{\geq}(X_1) = 1$. However, $\#_{\geq}(X_{11}) = 2$, as none of the conditions above hold. Clearly, all descendants of X_{11}, which are of type α or β, contain $indf(E)$ and either $indf(B)$ or $indf(A)$. In Fig. 5, Y_3 is a β-node, and so $\#_{\geq}(Y_3) = \#(Y_3.\omega) = 1$.

Note that the estimation presented above can be efficienty implemented by using two pre-compiled $|COMP| \times |COMP|$ matrices in which each entry (i, j) is a boolean value indicating (1) if c_i has a path to c_j in FDG and (2) if there is a component c_k which has pathes to both c_i and c_j. The matrices can be computed in $O(|COMP|^3)$.

Algorithm 3 outlines an improved algorithm which utilizes $\#_{\geq}(n)$. The generation of minimal diagnoses and the expansion of the DEG is incremental.

Algorithm 3: Computes all MDEs of a system
Input: A system
Output: $\Theta(\mathfrak{D})$
(1) If there is an empty diagnosis Δ: return $\Theta(\mathfrak{D}) := \{\Delta\}$
(2) For $\nu := 1, .., |COMP|$:
(3) Generate all minimal diagnoses Δ with $\#_{\geq}(\Delta) = \nu$; add them as root nodes to the DEG.
(4) Repeat
(5) Select an unexpanded α-node n s.t. $\#_{\geq}(n) = \nu$
(6) Create α-children of n (Alg. 1)
(7) until no more nodes are created
(8) For all n with $\#(n.\omega) = \nu$: create all descendants of type β (i.e., apply Alg. 2 repeatedly)
(9) $\Theta(\mathfrak{D}) := \{n.\omega \,|\, \#(n.\omega) = \nu \text{ and } \Upsilon \cup n.\omega \not\models \bot\}$
(10) If $\Theta(\mathfrak{D}) \neq \emptyset$: finish algorithm, return $\Theta(\mathfrak{D})$

Example 10 Suppose we have 2 minimal diagnoses, $\Delta_x = \{ab(B), ab(C)\}$ and $\Delta_z = \{ab(B), ab(E)\}$. The DEG tree for Δ_x is depicted in Fig. 4, the tree for Δ_z is not depicted. In the first iteration ($\nu = 1$), only Δ_x is generated, as $\#_{\geq}(\Delta_z) = 2$. The following α-nodes are created: X_1, X_2, X_3, X_8, and X_{11}. However, X_{11} is not expanded because $\#_{\geq}(X_{11}) = 2$. Then the β-nodes X_4 and X_5 are generated (line 8 in Alg. 3). Finally, consistency checks are performed for X_3, X_4, and X_5 (line 9). If at least one of these MAs is consistent with Υ, then the algorithm terminates. Otherwise, Δ_z is generated as well and added to the DEG as root node of a second tree. In the second iteration, all remaining nodes of the DEG are created.

A further improvement of the algorithm can be achieved by utilizing *conflict sets*. The following definition is similar to those in (Reiter 1987) and (de Kleer & Williams 1989):

Definition 11 (Conflict Set) *A* conflict set γ *is a partial MA with* $M(c) = dm(c) \cup \{ab(c)\}$ *s.t.* $\Upsilon \cup \gamma$ *is inconsistent.*

I.e., a conflict set may contain the literals $\neg ab(\cdot)$, $indf(\cdot)$, $df(\cdot, \cdot)$, and also $ab(\cdot)$ because SD may contain descriptions of abnormal behavior. We suppose that the reasoner which performs the consistency checks of $\Upsilon \cup n.\omega$ returns a conflict set, denoted by $n.\gamma$, if an inconsistency is detected. Note that $n.\gamma \subseteq n.\omega$.

The crucial insight is that it is often possible to infer from $n.\gamma$ that the MAs of all descendants of n must be inconsistent with Υ as well. Let $n.\tilde{\omega} \subseteq n.\omega$ denote a partial MA s.t. $n.\tilde{\omega} \subseteq n'.\omega$ for all descendants n' of n. In other words, $n.\tilde{\omega}$ contains all those modes of $n.\omega$ which are also contained in all descendants of n. Then, if $n.\gamma \subseteq n.\tilde{\omega}$, all descendants must be inconsistent with Υ, and we can prune the tree.

Therefore, Alg. 3 can be improved by performing the consistency checks for nodes n with $\#(n.\omega) = \nu$ already during the expansion of the DEG, and by omitting the expansion of a node n when it is clear that all descendants of n would be inconsistent with Υ. $n.\tilde{\omega}$ can be easily derived from the structure of the DEG and from the failure dependency graph.

Example 11 Suppose the consistency check for X_3 (Fig. 4) yields the conflict set $X_3.\gamma = \{indf(A), df(A, C), \neg ab(F)\}$. As $X_3.\tilde{\omega} = \{indf(A), df(A, B), df(A, C), \neg ab(E), \neg ab(F), \neg ab(G)\}$, X_3 is not expanded.

We also observe that the number of required consistency checks can be reduced by preserving computed conflict sets in a data structure (similar to the *nogood database* in an ATMS, see (de Kleer 1986)). Before performing a consistency check for a certain MA, we can search in this data structure for a conflict set which refutes this MA.

Example 12 Suppose, as in the previous example, that $X_3.\gamma = \{indf(A), df(A, C), \neg ab(F)\}$. Since $X_3.\gamma \subseteq X_8.\omega$, X_8 (and its descendants) must be inconsistent.

Finally, we present important properties of conflict sets:

Proposition 2 *If* γ *is a conflict set, then* $\sigma(\gamma, ab/indf)$ *and* $\sigma(\gamma, indf/ab)$ *are conflict sets, too.*

Proof. Suppose $ab(c) \in \gamma$. As $DMA \cup \{indf(c)\} \models ab(c)$, $\Upsilon \cup \sigma(\gamma, ab/indf)$ is inconsistent as well. Moreover, since the $indf$ predicate occurs only in DMA (see Def. 8), $\sigma(\gamma, indf/ab)$ must also be inconsistent. □

Proposition 3 *If* γ *is a conflict set and* $ab(c) \in \gamma$, *then* $\sigma(\gamma, ab(c)/df(\cdot, c))$ *is a conflict set, too.*

Discussion and Related Research

We proposed the concept of MDEs in order to tackle the issue of dependent failures in consistency-based diagnosis. We motivated our work using an example from the control system of an autonomous robot. We provided a formalization of our approach, and we discussed consequences of our definitions. Furthermore, we provided an algorithm which computes all MDEs of a system.

In case of dependent failures, the minimal diagnoses often do not contain all components which have failed. Thus, each MDE corresponds to a (in general not minimal) diagnosis, and MDEs reflect the causal order of failures. The latter is important as the order of failure often indicates the order in which components must be repaired.

The authors of (de Kleer, Mackworth, & Reiter 1992) propose to compute *kernel diagnoses* instead of minimal diagnoses. However, we hold the view that in many applications multiple failures are only likely if they are dependent failures. Therefore, we explicitely model these dependencies by means of the failure dependency graph, which captures all possible failure dependencies. This graph is used to focus the reasoning.

There is no general answer how a modeller can create the graph. In some applications it might be possible to compute the graph dynamically, based on a domain-specific model and on current observations. In this paper we assume that the modeller knows the possible failure dependencies.

In many cases the number of MDEs will be much smaller than the number of kernel diagnoses. In some cases, there may be even fewer MDEs than minimal diagnoses, because MDEs are cardinality minimal wrt $indf$ modes. However, the number of MDEs can still be daunting. Apart from the focusing strategies we outlined above, the description of abnormal behavior (in SD) and of behavior in case of dependent failures (in DFD) can significantly reduce the number of MDEs. For example, one could adopt the view that a component must be correct if it manifests only nominal behavior in the given observations (Raiman 1989).

The complexity of the computation of MDEs depends on different factors. The density of the failure dependency graph and the length of its pathes have a large impact. In general, our approach may be intractable if the cascade of dependent failures is not locally confined in a large system, i.e., if the dependent failures propagate throughout the entire system. Another issue is the fact that dependent failures can lead to minimal diagnoses with a high cardinality. Our algorithm addresses this problem by (incrementally) generating only those minimal diagnoses whose DEs may be minimal.

The author of (Lucas 2001) proposes a method for reasoning with uncertainty in MBD which enables stochastic dependencies among components. However, to the best of our knowledge our work is the first one within the framework of consistency-based MBD which tackles the issue of dependent failures at the level of logical reasoning.

The issue of dependent failures is somehow similar to the problem of faults in the structure of the system which lead to hidden interactions, e.g., the well-known bridge fault; see (Davis 1984), (Preist & Welham 1990) and (Böttcher 1995). In both cases a single *cause of failure* may lead to multiple-fault diagnoses. However, in case of structural faults there is a single *point of failure*, whereas in presence of dependent faults multiple components have actually failed.

The example at the beginning of this paper is from an autonomous robot. (Steinbauer & Wotawa 2005) and (Weber & Wotawa 2006) deal with diagnosis in autonomous robots; however, dependent failures are not considered.

An empirical evaluation of the performance of our algorithm will be part of our future work. Another open issue is the combination of our approach with behavioral modes (de Kleer & Williams 1989). Furthermore, future research should deal with the application of probabilistic focusing techniques.

References

Böttcher, C. 1995. No faults in structure? how to diagnose hidden interactions. In *IJCAI*, 1728–1735.

Davis, R. 1984. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence* 24:347–410.

de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

de Kleer, J., and Williams, B. C. 1989. Diagnosis with behavioral modes. In *Proceedings of the* 11^{th} *International Joint Conf. on Artificial Intelligence*, 1324–1330.

de Kleer, J.; Mackworth, A. K.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2–3):197–222.

de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28:127–162.

de Kleer, J. 1990. Using crude probability estimates to guide diagnosis. *Artificial Intelligence* 45:381–392.

Lucas, P. J. F. 2001. Bayesian model-based diagnosis. *International Journal of Approx. Reasoning* 27(2):99–119.

Preist, C., and Welham, B. 1990. Modelling bridge faults for diagnosis in electronic circuits. In *Proceedings of the First International Workshop on Principles of Diagnosis*.

Raiman, O. 1989. Diagnosis as trial - the alibi principle. In *International Model-Based Diagnosis Workshop*.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Steinbauer, G., and Wotawa, F. 2005. Detecting and locating faults in the control software of autonomous mobile robots. In *Proceedings of the* 19^{th} *International Joint Conf. on Artificial Intelligence*, 1742–1743.

Struss, P., and Dressler, O. 1989. Physical negation: Integrating fault models into the general diagnostic engine. In *Proceedings of the* 11^{th} *International Joint Conf. on Artificial Intelligence*, 1318–1323.

Weber, J., and Wotawa, F. 2006. Using AI techniques for fault localization in component-oriented software systems. In *Proceedings of the 5th Mexican International Conference on Artificial Intelligence (MICAI 2006)*.

Weber, J., and Wotawa, F. 2007. Diagnosing dependent failures in the hardware and software of mobile autonomous robots. In *Proceedings of the 20th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2007)*. To appear.

New results for Sensor Placement with Diagnosability Purpose

Abed Alrahim Yassine, Stéphane Ploix, Jean-Marie Flaus
Laboratoire des sciences pour la conception, l'optimisation et la production, G-SCOP
BP 46, Saint Martin d'Heres 38402, France
abed-alrahim.yassine@g-scop.inpg.fr, stephane.ploix@inpg.fr, jean-marie.flaus@inpg.fr

Abstract

Abstract: Maintenance and diagnosis of complex systems are common activities in the industrial world. Technological advances have led to a continuously increasing complexity of industrial systems. This complexity, which is due to an increasing number of components reduces in turn the reliability of plants. Therefore, fault diagnosis is becoming a growing field of interest. But fault diagnosis relies on sensors: efficient fault diagnosis procedures require a relevant sensor placement. This paper presents fundamental results for sensor placement based on diagnosability criteria. These results contribute to the design of sensor placement algorithms, which satisfies specifications composed of several sets: the components for which faults have to be isolable, the components for which faults have to be detectable but not necessarily isolable and the components for which faults have do not need to be detected.
Keywords: fault diagnosis, diagnosability, sensor placement

Introduction

Sensor placement decisions depend on expected objectives. For instance, in control theory, the sensor placement is used to provide sufficient information for the control of systems. Criteria deal with observability and controllability of the variables. (Madron & Veverka 1992) has proposed a sensor placement method which deals with linear system. This method makes use of the Gauss-Jordan elimination to find a minimum set of variables to be measured. This ensures the observability of variables while simultaneously minimizing the cost of sensors. In this theory, the observable variables include the measurable variables plus the unmeasured but deductible variables. Another method for sensor placement has been proposed in (Maquin, Luong, & Ragot 1997). This method aims at guaranteeing the detectability and isolability of sensor failures. The proposed method is based on the concept of redundancy degree in a variable and the structural analysis of the system model. The sensor placement can be solved with a matricial analysis of a cycle matrix or using the technique of mixed linear programming. (Commault, Dion, & Yacoub Agha 2006) has proposed a method of sensor location. In this method, they defined a new set of separators (Irreducible Input Separators), which generates sets of system variables in which additional sensors must be implemented to solve the considered problem.

However, in fault diagnosis, the goal of sensor placement should be to satisfy detectability and diagnosability properties. Detectability is the possibility of detecting a fault on a component and diagnosability is the possibility of identifying a fault on a component without this creating ambiguity with any other fault.

(Travé-Massuyès, Escobet, & Milne 2001) has proposed a method based on consecutive additions of sensors, which takes into account diagnosability criteria. The principle of this method is to analyze the physical model of a system from a structural point of view. This structural approach is based on Analytical Redundancy Relations (ARR) (Cassar & Staroswiecki 1997), which can be obtained from combinations of model constraints using bi-partite graph (Blanke *et al.* 2003) or elimination rules (Ploix, Désinde, & Touaf 2005), and on the corresponding signature table (Patton & Chen 1991). In a signature table, rows and columns represent respectively, the set of analytical redundancy relations and the set of considered faults. However, this method requires an a priori design of all the ARR for a given set of sensors. Unfortunately, up to now, no method has been able to guarantee to provide all the possible ARR.

This paper presents results for the design of sensor placement algorithms. Thanks to these results, the sensor placement satisfying diagnosability objectives becomes possible without designing ARR a priori. It is an important feature since it is no longer necessary to design all the possible ARR assuming all the variables are measured.

Problem formulation

In the following, the set of variables appearing in a constraint k is denoted: $var(k)$ and the set of variables appearing in the set of constraints K: $var(K) = \bigcup_{k \in K} var(k)$.

A system Σ can be described by a tuple (K_Σ, C_Σ). $var(K_\Sigma)$ is the set of variables that models observable phenomena influenced by Σ. The behavior is represented by constraints $K_\Sigma = \{\ldots, k_i, \ldots\}$ that establish relationships between variables of $var(K_\Sigma)$. It can be represented by a structural matrix $\mathcal{M}_\Sigma$, which is an incidence matrix

representing the application $\mathcal{M}_\Sigma : var(K_\Sigma) \rightarrow K_\Sigma$. $C_\Sigma = \{\ldots, c_j, \ldots\}$ is a set of independent components constituting Σ. Each constraint in K_Σ models one component and, conversely, a component can be modeled by at most one constraint: $\forall k \in K_\Sigma, comp(k) \in C_\Sigma$[1].

Let us introduce the concept of testable subsystem (TSS) and its relationship with the concept of ARR.

Definition 1 *A set of constraints is testable if all the constraints can be combined into at least one global constraint that no longer contains variables. The values appearing in the global constraint may correspond either to model parameters or to observations coming from measurements or from controlled variables.*

This definition also applies to models containing ordinary differential equations. Indeed, testable state space representations, including state space observers, always have equivalent parity space representations (Staroswiecki, Cocquempot, & Cassar May 5 11 1991).

Definition 2 *A testable set of constraints is minimal if it is not possible to keep testability when removing a constraint.*

A global testable constraint that can be deduced from a TSS is called ARR. Let $R_\Sigma = \{\ldots, r_k, \ldots\}$ be the set of all the testable subsystems that can be deduced from K_Σ according to (Blanke *et al.* 2003; Ploix, Désinde, & Touaf 2005; Staroswiecki & Declerck 1989).

Because of the one-to-one relationships between constraints and components, notions of detectability and discriminability can be extended to constraints.

Let R be a set of TSS coming from (K_Σ, C_Σ)[2].

Definition 3 *A constraint $k \in K_\Sigma$ is detectable (see (Struss* et al. *2002)) in R if $\exists r_i \in R / k \in r_i$. By extension, the constraints $K \subset K_\Sigma$ are detectable in R if $\forall k_i \in K$, k_i is detectable in R.*

Definition 4 *Two constraints $(k_1, k_2) \in K_\Sigma^2$ are discriminable (see (Struss* et al. *2002)) in R if: $\exists r_i \in R / k_1 \in r_i$ and $k_2 \notin r_i$. By extension, the constraints of a set $K \subset K_\Sigma$ are discriminable in R if: $\forall (k_i, k_j) \in K^2$, k_i and k_j are discriminable in R with $k_i \neq k_j$.*

Obviously, non detectability implies non discriminability.

Definition 5 *A constraint $k \in K_\Sigma$ is diagnosable (see (Struss* et al. *2002; Console, Picardi, & Ribando 2000)) in R if: it is detectable and if $\forall k_j \in (K_\Sigma \setminus k)$, (k, k_j) are discriminable in R. By extension, the constraints $K \in K_\Sigma$ are diagnosable in R if: $\forall k_i \in K$, k_i are diagnosable in R.*

In order to formulate the sensor placement problem, the notion of terminal constraint has to be introduced.

[1] A component may also be modeled by several constraints but, for the sake of simplicity, it has not been considered in this paper.

[2] C_Σ is not used at this stage.

Definition 6 *A terminal constraint k is a constraint that satisfies: $card(var(k)) = 1$ where $var(k)$ is the set of variables appearing in the constraint k. A terminal constraint usually models a sensor or an actuator. It is thus a major concept in sensor placement.*

In fault diagnosis, sensor placement has to satisfy specifications dealing with detectability and diagnosability. Because of the one-to-one relation between components and constraints, what is true for components is also true for constraints. Therefore, the components C_Σ and the corresponding constraints K_Σ may be decomposed into several sets:

- the set of components C_{diag} / constraints K_{diag} that has to be diagnosable
- the set of subsets of components $\mathbb{C}_{nondis} = \{\ldots, C_i, \ldots\}$ / constraints $\mathbb{K}_{nondis} = \{\ldots, K_i, \ldots\}$ that have to be non discriminable but detectable for each set C_i or K_i.
- the set of components C_{nondet} / constraints K_{nondet} that has to be non detectable

Specifications C_{diag}, $\mathbb{C}_{nondis}$ and C_{nondet} of sensor placement problems are meaningful if the two following properties are satisfied:

1. Sets in specifications must not to overlap one each other to make sense: constraint sets have to satisfy: $C_{nondet} \cap C_{diag} = \phi$, $\forall C_i \in \mathbb{C}_{nondis}, C_i \cap C_{nondet} = \phi$, $\forall C_i \in \mathbb{C}_{nondis}, C_i \cap C_{diag} = \phi$ and $\forall (C_i, C_j) \in \mathbb{C}_{nondis}^2$, $C_i \cap C_j = \phi$ if $C_i \neq C_j$ (no overlapping property).
2. The union of all the components appearing in C_{diag}, $\mathbb{C}_{nondis}$ and C_{nondet} has to correspond to C_Σ: $C_\Sigma = C_{diag} \cup C_{nondet} \cup \bigcup_{C_i \in \mathbb{C}_{nondis}} C_i$ (completeness property).

If these properties are satisfied the specifications are qualified as consistent in C_Σ . Replacing components by corresponding constraints leads to the same properties for specifications K_{diag}, $\mathbb{K}_{nondis}$ and K_{nondet} to be consistent in K_Σ.

Satisfying the specifications requires information delivered by sensors. Let Σ' represent the system Σ with the additional sensors. Σ' can be described by a tuple $(K_{\Sigma'}, C_{\Sigma'})$ where $C_{\Sigma'}$ represents the components of system Σ plus the additional sensors and $K_{\Sigma'}$ represents the constraints of system Σ plus the additional terminal constraints which model the sensors. The sensor placement problem consists in determining the additional terminal constraints in $K_{\Sigma'}$ that lead to the satisfaction of the specification K_{diag}, $\mathbb{K}_{nondis}$ and K_{nondet}. Because of the relations between constraints and components, the results can be extended to components.

In the next sections, fundamental results are proposed for the design of sensor placement satisfying diagnosability and detectability specifications. Algorithms are not detailed in this paper.

Preliminary concepts

Before deducing diagnosability properties of constraint sets, some concepts have to be introduced.

Value propagation as a theoretical tool

According to the definition, an TSS is a minimum set of constraints K such that there exists a constraint $k \in K$ for which all the variables of $var(k)$ can be instantiated, starting from terminal constraints. An ARR corresponding to a TSS can be seen in different ways. The most common approach is to consider an ARR as a constraint. Another way is to think of an ARR as a complete value propagation (Fron 1994) w.r.t. variables i.e. a propagation that leads to information about the consistency of a set of constraints, including terminal constraints that contain known data. This approach has been adopted as a theoretical tool to develop proofs. Relationship between value propagation and ARR is detailed in this section.

Let k_1 and k_2 be two constraints. The propagation of a variable v between k_1 and k_2 is possible only if $v \in var(k_1) \cap var(k_2)$. The variable v is qualified as propagable between k_1 and k_2. Consider a system, defined by $K_\Sigma = \{k_1, k_2, k_3, k_4, k_5\}$ with $var(k_1) = \{v_1, v_3\}$, $var(k_2) = \{v_1, v_2\}$, $var(k_3) = \{v_2, v_3\}$, $var(k_4) = \{v_2\}$ and $var(k_5) = \{v_3\}$. Terminal constraints k_4 and k_5 model sensors or actuators. Each terminal constraint contains known data. The set of all the tests that can be performed is represented by the propagations drawn in figure 1.

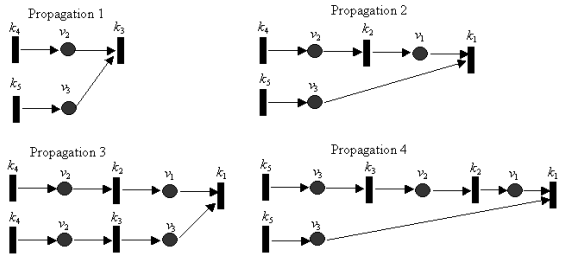


Figure 1: Set of propagation

A propagation starts by a terminal constraint, which means that "a variable is equal to a known value". In this example, propagations start either with k_4 or k_5. Thanks to these constraints, a value can be respectively assigned to v_2 and v_3. Once values have been assigned to these variables, new variables can then be instantiated. Propagation continues until no more assignments are possible because terminal constraints or instantiated variables have been reached. The set of constraints that appears in a propagation, corresponds to a testable subsystem. These constraints can be combined into a unique global constraint named ARR. Depending on the constraints chosen for propagating values, different ARR may be obtained (see figure 1).

In the continuation of this paper, value propagation is implicitly used and appears in the proofs of the different lemmas and theorems.

Some characteristics of constraint sets

The concept of *linked constraints* is introduced because it is important regarding sensor placement. Indeed, discriminability depends on this concept.

As mentioned in (Blanke *et al.* 2003), the constraints of a system Σ may be modeled by a non directed bipartite graph $(K_\Sigma, var(K_\Sigma), E_\Sigma)$ where E_Σ is the set of edges. Each edge $e = (k, v)$ models that $v \in var(k)$.
Let us introduce new definitions useful for sensor placement.

Definition 7 *A set of constraints $K \subset K_\Sigma$ is interconnected by a set of variables $V \subset var(K_\Sigma)$ iff there is a tree $(K, V, E) \subset (K_\Sigma, var(K_\Sigma), E_\Sigma)$ with constraints at extremities (see (Bollobás 1998) for example), which satisfies $card(V) = card(K) - 1$.*

Definition 8 *A set of constraints $K \subset K_\Sigma$ is linked in K_Σ by a set of variables $V \subseteq var(K_\Sigma)$ iff K is interconnected by V and iff the other constraints of K_Σ (i.e. $K_\Sigma \backslash K$) do not contain any variable of V. The variables of V are called linking variables for K. They are denoted: $var_{linking}(K, K_\Sigma)$.*

The shape of a structural matrix dealing with linked constraints is drawn in figure 2.

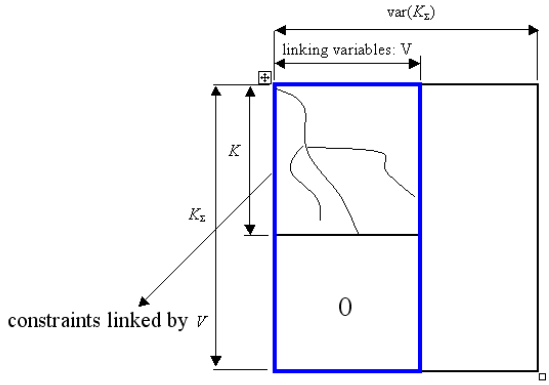


Figure 2: Structural matrix of a constraint set, which is linked by path

The concept of linked constraints is strongly connected with discriminability.

Lemma 1 *A set of constraints $K \subset K_\Sigma$ linked by a set of variables $V \subset V_\Sigma$ is necessarily non discriminable.*

proof 1 *Indeed,*

1. *because variables in V only appear in the constraints in K, the only way of propagating variables is to use the constraints in K and the variables in V,*

2. *because there is a tree $(K, V, E) \subset (K_\Sigma, var(K_\Sigma), E_\Sigma)$ with constraints at extremities, instantiating all the variables in V involves at least the achievement of the propagations defined by the tree.*

Therefore, all the constraints are invariably found together in TSS. In order to improve the clarity of these explanations, let us introduce the notion of stump variables.

Definition 9 *A set of variables $var(K)$ appearing in a set of constraints K but not in the other constraints of K_Σ (i.e. $K_\Sigma \backslash K$) are named stump variables in K_Σ. They are denoted: $var_{stump}(K, K_\Sigma)$.*

For instance, the set of variables V that links a set of constraints K belong to the stump variables $var_{stump}(K, K_\Sigma)$.

A set of constraints cannot be used to generate a TSS if they are linked and if there are additional variables that cannot be propagated. These constraints are qualified as isolated. Detectability depends on this concept.

Definition 10 *A set of several constraints $K \subset K_\Sigma$ is isolated in K_Σ by a set of variables $V \subset var(K_\Sigma)$ if they are linked by V and if there is at least one variable in $var(K) \backslash V$ that does not belong to other constraints of K_Σ (i.e. $K_\Sigma \backslash K$). If the set contains only one constraint, the link condition disappears but the other remains.*

The shape of a structural matrix dealing with isolated constraints is drawn in figure 3.

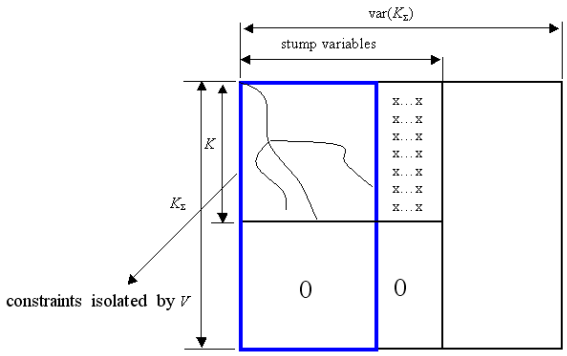


Figure 3: Structural matrix of a constraint set, which is isolated by the set of variables V

The concept of isolated constraints is strongly linked with detectability.

Lemma 2 *A set of constraints $K \subset K_\Sigma$ isolated in K_Σ by V is necessarily non detectable.*

proof 2 *The constraints K isolated in K_Σ by V will always come together in TSS because, by definition, they are linked by V. Because of the fact that, in isolated constraints, there is at least one additional variable in $var(K)$ which does not appear in other constraints (i.e. $K_\Sigma \backslash K$), it is not possible to instantiate this value and, therefore, this set of constraints cannot be involved into a TSS: K is non detectable.*

Constraint set and diagnosability properties

This section aims at setting up a direct link from sets of constraints to detectability and diagnosability properties. Firstly, it is obvious that adding additional constraints connected to all the variables $var(k)$ appearing in a constraint k, ensures the diagnosability of k.

Lemma 3 *Let $k \in K_\Sigma$ be a constraint. If additional terminal constraints dealing with all the variables in $var(k)$ are added, then the constraint k is diagnosable.*

proof 3 *Because there are additional terminal constraints connected to each variable in $V(k)$, a value can be assigned for each variable. Consequently, there is one TSS containing k plus additional terminal constraints connected to variables in $var(k)$. Therefore, the constraint $k \in K$ is necessarily diagnosable because there is one TSS that does not contain other constraints of K_Σ (i.e. $K_\Sigma \backslash \{k\}$).*

Lemma 3 can be directly applied to all the constraints of a constraint set.

corollary 1 *If additional terminal constraints dealing with all the variables $var(K)$ of a constraint set $K \in K_\Sigma$, then each constraint $k \in K$ is diagnosable.*

In lemma 2, a relationship between isolated constraints and the detectability property has been presented. The next lemma generalizes the previous results.

Lemma 4 *A sufficient condition for a subset of constraints $K \subset K_\Sigma$ to be non detectable is that there is a tuple $(K_1, \ldots, K_m)$ of m sets of constraints making up a partition $\mathcal{P}(K)$ of K such that each K_i is isolated in $K_\Sigma \backslash \bigcup_{j<i} K_j$ (K_1 is a limit case: it should be isolated in K_Σ) .*

proof 4 *The case of K_1 has been discussed in lemma 2: because the constraints in K_1 are isolated in K_Σ, they are non detectable and therefore cannot be included in TSS. Then, the remaining candidate constraints for TSS belong to $K_\Sigma \backslash K_1$. Because K_2 is isolated in $K_\Sigma \backslash K_1$, they are non detectable. The reasoning can be extended to any i. Consequently, the constraints in $K = \bigcup_i K_i$ are non detectable.*

Figure 4 indicates the shape of a structural matrix of non detectable constraints.

Consider, for example, a system modeled by the following structural matrix:

	v_1	v_2	v_3	v_4	v_5	v_6
k_1	1	0	0	1	0	0
k_2	0	1	1	0	1	0
k_3	0	1	1	0	1	0
k_4	0	0	0	1	0	1
k_5	0	0	0	1	1	1

Assume that the set $K = \{k_1, k_2, k_3\}$ is required to be non detectable. In this example, there exists a tuple $(\{k_1\}, \{k_2, k_3\})$ such that each element K_i satisfies lemma 4. If there are no additional terminal constraints containing v_1, v_2 and v_3, the subset K is non detectable.

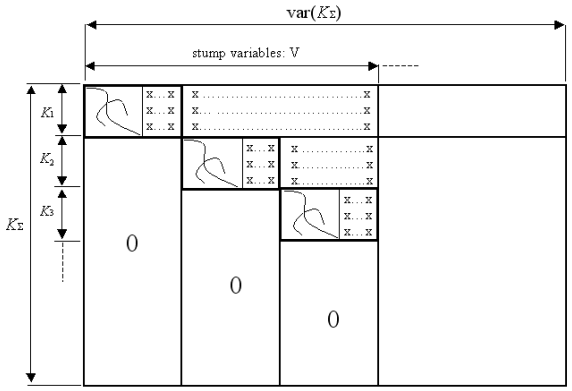


Figure 4: Structural matrix of non detectable constraints

Lemma 5 *A sufficient condition for each set $K_i \subset K$ belonging to a set of m constraint sets $\mathbb{K} = \{K_1, \ldots, K_m\}$ such that $\forall K_i \neq K_j, K_i \cap K_j = \emptyset$, to be non discriminable is that each K_i is linked by a set of variables V_i.*

proof 5 *This lemma is a direct application of lemma 1 to several sets of constraints.*

Consider, for example, a system modeled by the following structural matrix:

	v_1	v_2	v_3	v_4	v_5
k_1	1	0	1	1	1
k_2	1	1	1	1	0
k_3	1	1	1	0	1
k_4	0	1	1	0	0
k_5	0	0	0	1	1

Assume that $K = \{k_1, k_2, k_3, k_4\}$ is a constraint subset that should be non discriminable. Because the constraints k_1, k_2, k_3 and k_4 are linked by $V = \{v_1, v_2, v_3\}$, lemma 5 is satisfied. Therefore, k_1, k_2, k_3 and k_4 are non discriminable provided that no additional terminal constraints contain a variable of V.

The following theorem groups lemmas 3, 4 and 5.

theorem 1 *Let K_Σ be a set of constraints and K_{nondet}, $\mathbb{K}_{nondis}$ and K_{diag} be the specifications of a sensor placement problem consistent in K_Σ. Sufficient conditions for the specifications to be fulfilled are:*

1. *there exists a tuple $(K_1, \ldots, K_p)$ of p sets of constraints making up a partition $\mathcal{P}(K_{nondet})$ of K_{nondet} such that each K_i is isolated in $K_\Sigma \backslash \bigcup_{j<i} K_j$ (K_1 is a limit case: it should be isolated in K_Σ) see figure 4.*
2. *each set K_i belonging to $\mathbb{K}_{nondis} = \{K_1, \ldots, K_m\}$ such that $\forall K_i \neq K_j, K_i \cap K_j = \emptyset$, is linked by a set of variables V_i in considering only the constraints $K_\Sigma \backslash K_{nondet}$*
3. *Additional terminal constraints are added on the variables $V_{candidate} = var(K_\Sigma) \backslash (var_{stump}(K_{nondet}, K_\Sigma) \cup \bigcup_{K_j \in \mathbb{K}_{nondis}} var_{linking}(K_j, K_\Sigma \backslash K_{nondet}))$ (see figure 5).*

proof 6 *The proof relies on the resulting structure of the structural matrix, which directly stems from corollary 1 and lemmas 4 and 5. Note that point 2 could also be stated for the whole set of constraints K_Σ. However, it is not useful to include non detectable constraints, which will not appear in resulting TSS: it would be less conservative.*

Because of lemma 4 and 5, the variables of $var(K_{diag})$ cannot contain variables appearing in the variables involved in (1) and (2) i.e. in $var_{stump}(K_{nondet}, K_\Sigma)$ and in $\bigcup_{K_j \in \mathbb{K}_{nondis}} var_{linking}(K_j, K_\Sigma \backslash K_{nondet})$. Then, $var(K_{diag})$ satisfies: $var(K_{diag}) \subset V_{candidate}$. Because the variables of $V_{candidate}$ can be instantiated with measured values, all the constraints of K_{diag} are diagnosable following corollary 1.

The point that has to be proved is that, in specifications, $\mathbb{K}_{nondis}$ defines non discriminable but detectable sets and not only non discriminable sets as in lemma 5: the detectability of sets in $\mathbb{K}_{nondis}$ has to be proved.

The variables $var(K_i)$ of a constraint set $K_i \in \mathbb{K}_{nondis}$ can be decomposed into two sets: V_i^- and V_i^+ where $V_i^- = var_{linking}(K_i, K_\Sigma \backslash K_{nondet})$ contains the linking variables and V_i^+ contains the remaining variables $V_i^+ = var(K_i) \backslash V_i^-$. Because of lemma 4 and 5, the set V_i^+ cannot contain variables in $var_{stump}(K_{nondet}, K_\Sigma)$ and in $\bigcup_{K_j \in \mathbb{K}_{nondis}; K_j \neq K_i} var_{linking}(K_j, K_\Sigma)$. Therefore, V_i^+ satisfies: $V_i^+ \subset V_{candidate}$

Because of the third point of the theorem, all the variables of $V_{candidate}$ are known: additional terminal constraints are indeed added, there is necessarily a TSS dealing with all the constraints in K_i. It proves that the constraint set K_i is necessarily detectable. Because this result holds for any $K_i \in \mathbb{K}_{nondis}$, it proves the theorem.

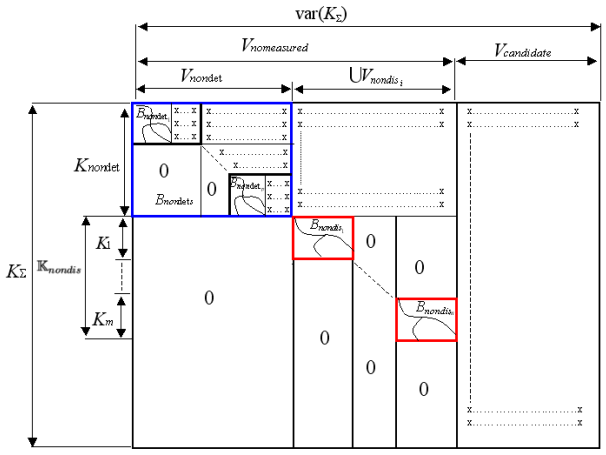


Figure 5: Shape of a structural matrix Satisfying theorem 1

Satisfying theorem 1 guarantees that the specifications are satisfied. However, because the theorem provides only a sufficient condition for diagnosability, the number of additional terminal constraints is not necessarily minimal.

It has to be checked afterwards.

The sensor placement problem has been studied without considering components. Let us now take components into account. Components of a system may be divided into three sets: the components on which faults need to be isolated, the components on which faults need to be detected but not necessarily localized and the components on which faults need to be non detectable. Because it has been assumed that each component is modeled by only one constraint, the results obtained for constraints can be extended to components using the application $\Phi_\Sigma : K_\Sigma \rightarrow C_\Sigma$.

Application to DAMADICS benchmark

Several methods for fault isolation have been benchmarked on a pneumatic servo-motor actuated valve named DAMADICS (Development and Application of Methods for Actuator diagnosis in Industrial Control Systems). (Spanache & Escobet 2004) has designed a sensor placement method for this problem that optimizes the diagnosability level of the system. In this section, the proposed results in this paper are applied on this benchmark in order to compute a sensor placement that satisfies diagnosability specifications.

The system is defined by the following equations:

$$\begin{aligned} k_1 &: \quad x = r_1(ps, \triangle p) \\ k_2 &: \quad fv = r_2(x, \triangle p) \\ k_3 &: \quad cvi = r_3(sp, pv) \\ k_4 &: \quad ps = r_4(x, cvi, pz) \\ k_5 &: \quad pv = r_5(x) \end{aligned}$$

The corresponding structural matrix is given by:

	x	ps	cvi	pv	fv	pz	sp	$\triangle p$
k_1	1	1	0	0	0	0	0	1
k_2	1	0	0	0	1	0	0	1
k_3	0	0	1	1	0	0	1	0
k_4	1	1	1	0	0	1	0	0
k_5	1	0	0	1	0	0	0	0

Let's fix these specifications: $K_{nondet} = \{k_4\}$, $\mathbb{K}_{nondis} = \{\{k_1, k_2\}, \{k_3, k_5\}\}$ and $K_{diag} = \{\phi\}$.

The constraint k_4 is isolated by $\{pz\}$, then the lemma 4 is satisfied.

The set of constraints $K_1 = \{k_1, k_2\} \in \mathbb{K}_{nondis}$ is linked by $\{\Delta p\}$, and the set of constraints $K_2 = \{k_3, k_5\} \in \mathbb{K}_{nondis}$ is linked by $\{pv\}$.

According to theorem 1, no terminal constraints containing a variable from $\{pz, \triangle p, pv\}$ have to be added i.e. these variables must not be measured.

In order to satisfy the last item of theorem 1, all the variables of the system except $\{pz, \triangle p, pv\}$ have to be measured. These measured variables $V_{candidate} = \{x, ps, cvi, fv, sp\}$ represent the candidate variables.

The method proposed in (Ploix, Désinde, & Touaf 2005) has been used to design all the ARR. It has led to this fault signature table:

ARR	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}
ARR_1	0	0	1	0	1	1	0	1	0	1
ARR_2	1	1	1	0	1	0	1	1	1	1
ARR_3	1	1	0	0	0	1	1	0	1	0

According to these results, the constraint sets that cannot be discriminated is: $\{k_1, k_2\}$ and $\{k_3, k_5\}$ and the constraint set that cannot be detected is: $\{k_4\}$. Applying the function $\Phi : K_\Sigma \longrightarrow C_\Sigma$, it is obvious that the components that cannot be discriminated are: $\{c_1, c_2\}$ and $\{c_3, c_5\}$ and the component that cannot be detected is: $\{c_4\}$.

Suppose now that the specifications are given by: $\mathbb{K}_{nondis} = \{\{k_3, k_4, k_5\}\}$ and $K_{diag} = \{k_1, k_2\}$.

The set of constraints $\{k_3, k_4, k_5\} \in \mathbb{K}_{nondis}$ is linked by $\{cvi, pv\}$.

According to theorem 1, no terminal constraints containing a variable from $\{cvi, pv\}$ have to be added i.e. these variables must not be measured.

In order to satisfy the last item of theorem 1, all the variables of the system except $\{cvi, pv\}$ have been measured. These measured variables $V_{candidate} = \{x, ps, fv, pz, sp, \triangle p\}$ represent the candidate variable.

The proposed method in (Ploix, Désinde, & Touaf 2005) has been used to produce all the TSS. It leads to the following fault signature matrix:

ARR	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}
ARR_1	0	0	1	1	1	1	1	0	1	1	0
ARR_2	1	1	1	1	1	0	1	1	1	1	0
ARR_3	1	0	1	1	1	0	1	0	1	1	1
ARR_4	1	1	0	0	0	1	1	1	0	0	0
ARR_5	1	1	1	1	1	1	0	1	1	1	0
ARR_6	1	0	0	0	0	1	1	0	0	0	1
ARR_7	0	1	1	1	1	0	1	1	1	1	1
ARR_8	1	1	1	1	1	0	0	1	1	1	1
ARR_9	1	1	0	0	0	0	1	1	0	0	1
ARR_{10}	0	1	0	0	0	1	0	1	0	0	1
ARR_{11}	1	0	1	1	1	1	0	0	1	1	1

According to these results, the constraint set that cannot be discriminated is: $\{k_3, k_4, k_5\}$ and the diagnosable constraint set is: $\{k_1, k_2\}$. Applying the function $\Phi : K_\Sigma \longrightarrow C_\Sigma$, it is obvious that the components that cannot be discriminated are: $\{c_3, c_4, c_5\}$ and the diagnosable component set is: $\{c_1, c_2\}$.

Suppose that the specifications are given by: $\mathbb{K}_{nondis} = \{\{k_3, k_4\}, \{k_2, k_5\}\}$ and $K_{diag} = \{k_1\}$.

The set of constraints $K_1 = \{k_3, k_4\} \in \mathbb{K}_{nondis}$ is linked by $\{cvi\}$, but the set of constraints $K_2 = \{k_2, k_5\} \in \mathbb{K}_{nondis}$ cannot be linked by a set of variables. Consequently, theorem 1 is not satisfied and there is no solution that satisfies these specifications.

The results presented in this paper demonstrate that it is possible to design sensor placements which satisfy diagnosability criteria without designing ARR a priori.

Conclusion

New results for the design of sensor placement algorithms has been proposed. It manages, the specifications dealing with sets of constraints that have to be diagnosable, non discriminable or non detectable. These results apply to any system depicted by constraints, which may only be described by the variables appearing in them. Thanks to these results, sensor placements satisfying diagnosability specifications become possible without designing ARR a priori. It is a very important feature since it is no longer necessary to design all the possible ARR assuming that some variables are measured. An algorithm providing solutions to the sensor placement problem that contains a minimum number of sensors will be provided in the near future.

References

Blanke, M.; Kinnaert, M.; Lunze, J.; and Staroswiecky, M. 2003. *Diagnosis and fault tolerant control*. Springer-Verlag.

Bollobás, B. 1998. *Modern graph theory*. Graduate Texts in Mathematics. New York, U.S.A.: Springer.

Cassar, J., and Staroswiecki, M. 1997. A structural approach for the design of failure detection and identification systems. In *IFAC, IFIP,IMACS Conference on control of industrial Systems*, 329–334.

Commault, C.; Dion, J.-M.; and Yacoub Agha, S. 2006. Structural analysis for the sensor location problem in fault detection and isolation. In *SAFEPROCESS'2006*.

Console, L.; Picardi, C.; and Ribando, M. 2000. Diagnosis and diagnosability analysis using process algebra. In *DX'2000*.

Fron, A. 1994. *Propagation par contraintes*. Addison-Wesley.

Madron, F., and Veverka, V. 1992. Optimal selection of measuring points in complex plants by lineair models. *AICheE* 38(2):227–236.

Maquin, D.; Luong, M.; and Ragot, J. 1997. Fault detection and isolation and sensor network design. *European Journal of Automation* 31(2):393–406.

Patton, R., and Chen, J. 1991. A review of parity space approaches to fault diagnosis. In *IFAC SAFEPROCESS Symposium*.

Ploix, S.; Désinde, M.; and Touaf, S. 2005. Automatic design of detection tests in complex dynamic systems. In *16th IFAC World Congress*.

Spanache, S., and Escobet, T. 2004. Fault diagnosability: a component-oriented approach. In *5th DAMADICS Workshop*.

Staroswiecki, M., and Declerck, P. 1989. Analytical redundancy in non-linear interconnected systems by means of structural analysis. In *IFAC Advanced Information Processing in Automatic Control (AIPAC'89)*, 51–55.

Staroswiecki, M.; Cocquempot, V.; and Cassar, J. May 5-11, 1991. Observer based and parity space approaches for failure detection and identification. In *IMACS-IFAC International Symposium*, 536–541.

Struss, P.; Rehfus, B.; Brignolo, R.; Cascio, F.; Console, L.; Dague, P.; Dubois, P.; Dressler, O.; and Millet, D. 2002. Model-based tools for the integration of design and diagnosis into a common process- a project report. In *DX'02*.

Travé-Massuyès, L.; Escobet, T.; and Milne, R. 2001. Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem. In *12th Int,Workshop on principles of diagnosis*, 205–212.

Finding Explanations in Bayesian Networks

Changhe Yuan
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
cyuan@cse.msstate.edu

Tsai-Ching Lu
HRL Laboratories, LLC
Malibu, CA, 90265
tlu@hrl.com

Abstract

Maximum a Posteriori (MAP) and Most Probable Explanation (MPE) are popular approaches of finding explanations for given observations in Bayesian networks. One of their shortcomings is that they have to find a complete assignment to a set of target variables. However, it is often the case that only few of the target variables are most relevant in explaining the observations. In this paper, we formulate the problem of finding the most relevant variables as *explanatory* MAP (eMAP), which considers all subsets of the target variables and finds a partial assignment that maximizes a chosen quality measure. We consider two quality measures for eMAP: Bayes Factor (or Weight of Evidence) and likelihood function. We then illustrate the proposed methodology on a circuit diagnosis problem in literature.

Introduction

Bayesian networks (BNs) (Pearl 1988) offer a compact and intuitive graphical representation of uncertain relationships among random variables in a domain. They have proven their value in many disciplines during the last two decades, including a variety of decision problems in medical diagnosis, prognosis, therapy planning, machine diagnosis, user modeling, natural language interpretation, planning, vision, robotics, data mining, fraud detection, and many others. Bayesian networks provide principled approaches for finding the most likely state of a system given new observations, typically formulated as Maximum a Posteriori (MAP) or Most Probable Explanation (MPE) problems. MAP is defined as follows. Let $\mathbf{X}$ be the set of MAP nodes whose most probable configuration is of our interest. Let $\mathbf{E}$ be the set of evidence nodes and $\mathbf{e}$ be their states. The remainder of the nodes is denoted by $\mathbf{Y}$. Then, the standard MAP query $MAP(\mathbf{X}, \mathbf{e})$ is defined as

$$MAP(\mathbf{X}, \mathbf{e}) \triangleq \hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \sum_{\mathbf{y}} P(\mathbf{X}, \mathbf{Y}|\mathbf{e}). \quad (1)$$

MPE is defined analogously except that $\mathbf{Y}$ is empty. In other words, MPE is interested in all the unobserved variables. Both MAP and MPE return a complete assignment to $\mathbf{X}$ as the best explanation for the evidence. In diagnostic setting, $\mathbf{X}$ typically contains fault variables in a system. The number of fault variables can be as large as several hundreds in a large model. In such case, even the best solution by MAP or MPE may have a very low probability. It is doubtful whether such an unlikely event constitutes a good explanation.

In a real world problem, it is often the case that only few of the target variables are most relevant in explaining the evidence. Take a medical domain as an example. When a patient goes to the hospital with certain symptoms, a physician can typically identify the only few diseases that the patient may have. All other diseases are regarded as unrelevant and ruled out from further consideration. Expert knowledge in this case helps in making the initial diagnosis. The question is whether we can formulate the problem formally so that we can develop computational solutions for solving it.

In this paper, we formulate the problem of identifying the most relevant target variables as *explanatory* MAP (eMAP), which considers all subsets of the target variables and finds a partial assignment that maximizes a chosen quality measure. Instead of restricting to any single measure, we compare two choices: Bayes factor (or Weight of Evidence) and likelihood function. We illustrate the method on circuit diagnosis problem in literature by comparing against several existing approaches.

Related Work

Besides MAP and MPE, there are several other existing approaches for finding explanations in Bayesian networks. We briefly review some of the approaches in this section.

Shimony (1993) defines explanations in Bayesian networks as the most probable independence-based assignment that is complete and consistent with respect to the evidence nodes. Roughly speaking, an explanation is a truth assignment to the variables "relevant" to evidence nodes. Only ancestors of evidence nodes can be relevant. An ancestor of a given node is irrelevant if it is independent of that node given the values of other ancestors.

Henrion and Druzdzel (1991) assume that the system has a set of pre-specified scenarios as potential explanations. Although they do not require the truth values of all propositions be specified, they order the explanations based on the posterior probability of the scenarios.

de Campos et al. (de Campos, Gamez, & Moral 2001) proposed yet another approach for finding explanations in

Bayesian networks. Their approach is a two-step procedure. They first solve a K-MPE problem and find the K most probable explanations in a Bayesian network given the evidence. Then, they simplify the explanations by greedily removing unimportant variables one by one. If the removal of a literal does not reduce the likelihood of an explanation or only reduce the likelihood within a certain factor, the literal is regarded as unimportant and can be removed from the explanation.

There are also some approaches that are not specific to Bayesian networks. Gärdenfors (1988) defines that X is an explanation of E relative to a state of belief $K =< W, P >$, where W is a set of possible worlds and P is a distribution on W and $E \in K$, if

1. $P_E^-(E|X) > P_E^-(E)$, and
2. $P(X) < 1 (i.e., X \notin K)$.

Gärdenfors orders all legitimate explanations using *explanatory power* (EP). The EP of X with respect to E is defined as

$$EP(X,E) = \frac{P_E^-(E|X)}{P_E^(E)}. \quad (2)$$

Since $P_E^-(E)$ is a constant for given E, EP essentially orders the explanations according to the likelihood of evidence given the explanations.

Chajewska and Halpern (1997) assume that a world is a pair (w, C) consisting of a truth assignment w and a causal structure C. They also assume that the explanandum E is one of the observations O and define P_E^- as follows:

$$P_E^-(w,C) = P'(C)P_C(w|O - \{E\}). \quad (3)$$

They place only partial orderings on explanations based on a pair of numbers $(EP(X,E), P_E^-(X))$: the explanatory power and the prior of X.

All these approaches realize the need to find variables that are most relevant to the evidence. However, as we will show later, they may still find explanations that contain too many variables, some of which may be unimportant. We propose a new approach for the problem in the next section.

Explanatory MAP

Based on the assumption that typically only few of the target variables are most relevant in explaining the evidence in a system, we formulate identifying the most relevant fault variables as *explanatory* MAP query $eMAP(\mathbf{X}, \mathbf{e})$ which takes the same input as MAP problem: the set of MAP nodes $\mathbf{X}$, the set of evidence nodes $\mathbf{E}$, and the remaining nodes $\mathbf{Y}$, and solves the following optimization problem.

$$eMAP(\mathbf{X}, \mathbf{e}) \triangleq \hat{\mathbf{x_k}} = \arg\max_{\mathbf{x_k}, \mathbf{X_k} \subseteq \mathbf{X}} f(\mathbf{x_k}, \mathbf{e}), \quad (4)$$

where $f(\mathbf{x_k}, \mathbf{e})$ is an evaluation function that measures the degree of a partial assignment to K variables $\mathbf{X_k}$ in $\mathbf{X}$ in explaining $\mathbf{e}$. In other words, we are searching for a solution simultaneously over the subspaces of potential MAP variables and their corresponding configurations. Suppose there are n MAP variables and each has d states, the size of the search space of eMAP would be $\sum_{i=1}^{n} \binom{n}{i} d^i$, which is much larger than d^n of MAP.

One element missing from the formulation is the evaluation function f. A naive selection for f is the posterior probability of the partial assignment, which is obviously not a viable choice due to the fact that the probabilities in comparisons are not at the same footing. Assignments with fewer variables always dominate those with more variables. In this paper, we compare two quality measures: likelihood function and Bayes factor (Good 1985). Both likelihood and Bayes factor provide "common" grounds for comparing all the partial assignments of $\mathbf{X}$.

Likelihood Function

The first choice for f is the likelihood function $P(\mathbf{e}|\mathbf{X_k})$. By using this measure, we basically formulate eMAP as a model selection problem: Each partial assignment of the MAP variables serves as a model candidate, and we want to find the model that mostly likely generates the data, in our case, the evidence. Since adding variables that are conditionally independent of evidence to an explanation does not reduce the likelihood of the explanation, we should penalize more complex explanations as in model selection. We introduce a penalty term as in AIC (Akaike 1974) for eMAP. In the general case, AIC is defined as follows:

$$AIC = -2 \ln L(M) + 2K(M),$$

where $L(M)$ is the likelihood of the data given model M and $K(M)$ is the number of parameters. It is equivalent to using $P(\mathbf{e}|\mathbf{X_k})exp(-K)$ as the quality measure in eMAP, where K is the number of variables. $P(\mathbf{e}|\mathbf{X_k})$ is the likelihood, and $exp(-K)$ is an exponential prior penalizing the complexity of the model. However, when K is small, change in $exp(-K)$ when K changes may dominate the whole measure. In order to minimize the effect of the prior, we use $exp(-cK)$ instead, where c is small constant far less than 1. This prior is only intended to help discriminating explanations with different number of variables when their likelihood scores are similar.

Bayes Factor

Another choice for f in Eqn. 4 is Bayes factor (BF) (Good 1985). Suppose we have data D assumed to have arisen under one of the two hypotheses H_1 and H_2 according to a probability density $P(D|H_1)$ and $P(D|H_2)$. The Bayesian factor is given by

$$BF = \frac{P(D|H_1)}{P(D|H_2)}. \quad (5)$$

Bayes factor is similar to likelihood ratio, although in Bayesian setting we have to average likelihood over the parameters. The logarithm of BF is called the *Weight of Evidence* (WoE) given by D for H_1 over H_2 (Good 1985). A value of $BF > 1$ means that the data indicate that H_1 is more likely than H_2. Jeffreys (1961) gave the following scale for interpretation of BF:

In our problem, there are many explanations that compete with each other, so we use a generalized version of Bayes

BF	Strength of evidence
$<$ 1:1	Negative (supports H_2)
1:1 to 3:1	Barely worth mentioning
3:1 to 12:1	Positive
12:1 to 150:1	Strong
$>$ 150:1	Very strong

Table 1: Interpretation of Bayes factor (BF).

factor (Fitelson 2001)

$$BF = \frac{P(D|H_1)}{P(D|\neg H_1)}. \quad (6)$$

Using Bayes rule, we can transform the above definition into the following form:

$$BF = \frac{P(H_1|D)(1-P(H_1))}{P(H_1)(1-P(H_1|D))}. \quad (7)$$

In our problem, D is simply the evidence $\mathbf{e}$, and Hs are the explanations $\mathbf{x_k}$. We can use inference algorithms to get both $P(\mathbf{x_k})$ and $P(\mathbf{x_k}|\mathbf{e})$ in a Bayesian network.

Example

We illustrate our method using a concrete example, which originally appeared in (Poole & Provan 1991). We also compare our explanations against those of several existing approaches.

The Model

Consider a simple circuit as in Figure 1 (Poole & Provan 1991). A, B, C and D are gates that are faulty if they are closed, and the prior probabilities that the gates break independently are as follows:

$$P(A) = 0.016;$$
$$P(B) = 0.1;$$
$$P(C) = 0.15;$$
$$P(D) = 0.1.$$

We observe that there is current flowing through the circuit.

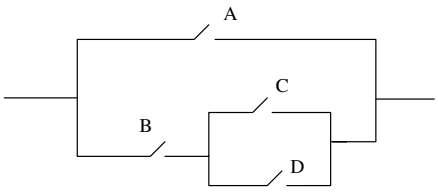


Figure 1: A circuit.

The circuit can be translated into an equivalent Bayesian network model as in Figure 2. Nodes A, B, C and D correspond to the gates in the circuit and each has two states "defective" and "ok". Node $Current$ represents the input to the circuit, with two states "yes" and "no". Other nodes are intermediate or final outputs, which can take state "current" or "ok". An intermediate output node takes state "current" if and only if one of its parent is in state "current" and the gate preceding it is "defective". The $Total\ Output$ node takes "current" if any of its parents is in state "current". The observation that current flows through the circuit is equivalent to the evidence that node $Current$ is in state "yes" and node $Total\ Ouput$ is in state "current". Given the evidence, the task is to diagnose the system and find the best explanations for the observations.

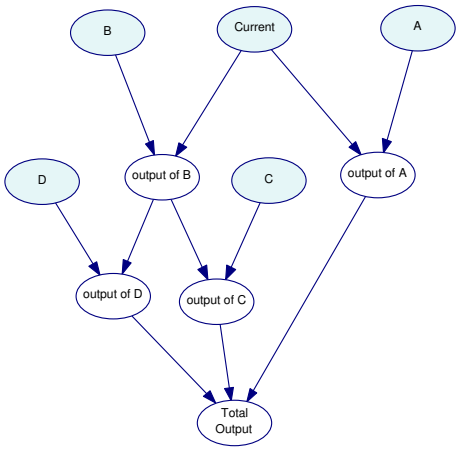


Figure 2: An equivalent BN model.

Based on our knowledge of the system, there are three scenarios which may lead to the observation:

1. A is defective;
2. B and C are defective;
3. B and D are defective;

Of course, any combination of the above three scenarios can also produce the observation. But the above three explanations are *minimal* in the sense that any explanation subsumed by them cannot fully explain the evidence. In the following sections, we compare the explanations found by multiple approaches.

To simplify notation in later discussions, we use OX to stand for variable $Ouput\ of\ X$, e.g., OA for $Output\ of\ A$. TO stands for $Total\ Output$. We use a variable and its negation to stand for the variable's two states.

Explanations by MPE and MAP

Most Probable Explanation (MPE) returns the most likely configuration of all the unobserved nodes as the explanation. The best explanation by MPE for the example is

$$P(\neg A \wedge B \wedge C \wedge \neg D \wedge \neg OA \wedge OB \wedge OC \wedge \neg OD) = 0.3395.$$

The MPE solution subsumes one of the minimal explanations asserting that B and C are defective. However,

OA, OB, OC and OD are not really necessary for the explanation, because removing them still results in a valid explanation.

The single MPE solution may not provide enough insight to the problem of a system. This problem can be addressed by solving the K-MPE problem that finds the K most probable explanations. If we set K equal to 3, K-MPE will output the following two other likely explanations.

$$P(A \wedge \neg B \wedge \neg C \wedge \neg D \wedge OA \wedge \neg OB \wedge \neg OC \wedge \neg OD) = 0.2816;$$

$$P(\neg A \wedge B \wedge \neg C \wedge D \wedge \neg OA \wedge OB \wedge \neg OC \wedge OD) = 0.2138.$$

In comparison, Maximum a Posteriori (MAP) focuses on the nodes that may be faulty. In the example model, A, B, C and D are the fault variables. The best explanation returned by MAP is

$$P(\neg A \wedge B \wedge C \wedge \neg D) = 0.3395.$$

Similarly, by allowing finding K-MAP solutions, we can also find two other likely explanations:

$$P(A \wedge \neg B \wedge \neg C \wedge \neg D) = 0.2816;$$
$$P(\neg A \wedge B \wedge \neg C \wedge D) = 0.2138.$$

MAP explanations are much more concise than those of MPE. However, for a large system with tens or hundreds of fault variables, even MAP solutions would contain too many variables. Some of the variables may not be necessary for the validity of the explanations.

Explanations By Shimony's Approach

Based on the definition of "relevance" in (Shimony 1993), $(\neg A, \neg OA, \neg TO)$ is an independence-based assignment with regard to evidence $(\neg Current, \neg TO)$, because once OA is observed at state "current", OC or OD become context specific independent (Boutilier *et al.* 1996) of TO. Such independence does not result from d-separation (Pearl 1988), which is the inherent property of the graphical structure, but rather the parameterizations of the model. Similarly, $(\neg B, \neg OB, \neg C, \neg OC, \neg TO)$ and $(\neg B, \neg OB, \neg D, \neg OD, \neg TO)$ are two other independence-based assignments. The probabilities of these three assignments are as follows:

$$P(\neg A, \neg OA, \neg TO) = 0.016;$$
$$P(\neg B, \neg OB, \neg C, \neg OC, \neg TO) = 0.015;$$
$$P(\neg B, \neg OB, \neg D, \neg OD, \neg TO) = 0.010.$$

Therefore, $(\neg A, \neg OA, \neg TO)$ is the independence-based MAP assignment in Shimony's approach.

This approach also finds an explanation that subsumes one of the three minimal explanations. Since it uses posterior probability to rank the explanations, it biases towards explanations with fewer variables and finds a more concise explanation than MPE and MAP. The drawback of this approach is that its solution may still contain unnecessary variables. As pointed out in (Chajewska & Halpern 1997), for each evidence node, the explanation must include an assignment to all the nodes in at least one path from the evidence to the root, because for each relevant node, at least one of its parents must be relevant. Furthermore, the irrelevance condition is quite strong and only achieves significant pruning in limited contexts.

Explanations by de Campos et al.'s Approach

de Campos et al.'s approach first finds the top K MPE solutions and then simplify the explanations by removing variables that do not or only slightly reduce the likelihood of evidence given the explanations. The top three MPE explanations are already listed in a previous section. It turns out that the likelihood of evidence is 1 given any of the three explanations.

After simplifying the explanations, de Campos et al.'s approach will obtain the following three explanations, which turns out to be the three minimal scenarios listed before.

$$P(e|\neg A) = 1;$$
$$P(e|\neg B, \neg C) = 1;$$
$$P(e|\neg B, \neg D) = 1.$$

One drawback of this approach is that likelihood does not distinguish among the above three explanations and treat them as equally good. Another drawback is that $P(e|X)$ does not always change monotonically as we remove variables from an explanation. Therefore, the simplifying procedure may be stuck in a local maximum. de Campos et al. recommend relaxing the criteria and allow $P(e|X)$ reduce by a small factor at each step. Another drawback, as the authors pointed out themselves, simplifying the explanations may result in fewer than K explanations.

Explanations by Gärdenfors' Approach

The probability of the evidence in this example equals 0.0391. There are many explanations that have likelihood greater than this value. Indeed, the likelihood of evidence is 1 given any of the three minimal explanations and any of the explanations subsuming one of the three minimal explanations. Since Gärdenfors orders the explanations essentially in the order of likelihood, all these explanations are regarded as equally good based on Gärdenfors' approach.

Explanations by eMAP

Finally, let us look at explanations by eMAP. We consider two quality measures separately: likelihood function and Bayes factor.

If we use likelihood function as the quality measure, eMAP will also find many explanations with the same quality, including the three minimal explanations and all explanations subsuming them. As we discussed before, we address the problem by introducing prior $P(K) \propto exp(-cK)$ to bias towards simpler models. eMAP with this prior would identify $P(e|\neg A) = 1$ as the top choice and the following

explanations slightly worse.

$$P(e|\neg B, \neg C) = 1;$$
$$P(e|\neg B, \neg D) = 1;$$
$$P(e|\neg A, B) = 1;$$
$$P(e|\neg A, \neg B) = 1;$$
$$P(e|\neg A, C) = 1;$$
$$P(e|\neg A, \neg C) = 1;$$
$$P(e|\neg A, D) = 1;$$
$$P(e|\neg A, \neg D) = 1.$$

Likelihood provides no further ability to discriminate between these explanations because they all consist of the same number of variables.

Now, let us consider Bayes factor. The major difference between Bayes factor and likelihood function is that Bayes factor not only factors in the ratio between the posterior and prior probabilities but also the magnitudes of the probabilities. When using Bayes factor as the quality measure, eMAP returns the following top explanation.

$$BF(\neg A) = 42.56;$$

The explanations immediately following the above explanation are:

$$BF(\neg B, \neg C) = 40.82;$$
$$BF(A, \neg B, \neg C) = 40.43;$$
$$BF(\neg A, D) = 39.87;$$
$$BF(\neg A, B) = 39.87;$$
$$BF(\neg A, C) = 38.65;$$
$$BF(\neg B, \neg C, D) = 38.5;$$
$$\ldots$$
$$BF(\neg B, \neg D) = 33.99;$$
$$\ldots$$

According to Jeffreys' interpretation of BF, these explanations all enjoy strong support from the evidence. Explanation $(\neg A)$ ranks on the top because its prior and posterior probabilities are both high. Explanation $(\neg B, \neg D)$ does not rank very high in the list due to its low prior probability. In other words, Bayes factor factors in both prior and posteriori probabilities such that explanations which cannot be distinguished by likelihood measure can be ranked using Bayes factor. This is especially important in sequential diagnosis as illustrated in the next section.

Sequential Diagnosis

In sequential diagnosis, we are normally presented with some initial evidence for a defective system and start ranking diagnostic hypotheses according to how well they can explain the evidence. Next, we rank tests based on their ability to differentiate a set of focus hypotheses, so called focus faults in (Lu & Przytula 2006). We then gather more evidence by performing the selected tests to refine our hypotheses. We typically go through several iterations of information gathering and hypotheses refining until we are confident enough to conclude our final diagnosis.

For our example, given the initial evidence, we have discussed the initial ranking of hypotheses for each method in the earlier sections. In this section, we assume that our focus hypothesis will be the top hypothesis only. We also assume that we have a silver-bullet test for verifying the health of A and the test is ranked as the top test to perform. If the test result confirms that A is okay, i.e. $\Pr(A) = 1$, we can include $A =$ "*ok*" as our additional evidence and continue the diagnosis.

We first look at eMAP with Bayes factor measure. In the first iteration, $(\neg A)$ would be our diagnosis hypothesis. If our test results show that indeed A is faulty, we found the problem of the system. Otherwise if A is okay, we have to include the fact that A is okay as our additional evidence and continue the diagnosis. In the next step, we found the following top explanations

$$BF(\neg B, \neg C) = 115.88;$$
$$BF(\neg B, \neg C, D) = 98.65;$$
$$BF(\neg B, \neg D) = 73.33;$$
$$BF(\neg B, C, \neg D) = 66.1;$$
$$BF(\neg B, \neg C, \neg D) = 45.39.$$

Therefore, $(\neg B, \neg C)$ is our next hypothesis to test.

If we use likelihood function as the quality measure for eMAP, the first step is the same. We will test hypothesis $(\neg A)$ first because of its simplicity. If A is okay, the next step is a little problematic because likelihood cannot distinguish between explanations $(\neg B, \neg C)$ and $(\neg B, \neg D)$. Other approaches using likelihood discussed above have the same problem.

It is possible to use Most Probable Explanations (MPE) or Maximum a Posteriori (MAP) Assignments to generate multiple-fault hypotheses. Both MPE and MAP ranks the explanations containing $(\neg B, \neg C)$ as their top choice. Therefore, we need to test this hypothesis first. However, it is typically harder to test hypotheses with more faults. For comparison purpose, we assume users choose to override the recommendation and test hypothesis $(\neg A)$ first. If A is okay, the following top explanations will be generated.

$$P(\neg B \wedge \neg C \wedge D) = 0.5745;$$
$$P(\neg B \wedge C \wedge \neg D) = 0.3617;$$
$$P(\neg B \wedge \neg C \wedge \neg D) = 0.0638.$$

Shimony's approach works pretty well for this example. No matter what results testing hypothesis containing $\neg A$ yields, the ordering of the other two top explanation will not change because they are independent from the first hypothesis.

Other Applications of eMAP

eMAP has the capability to automatically identify the variables that are most relevant in explaining new observations. It has potentially a wide range of applications in diagnostic settings. Besides generating multiple-fault hypotheses, we discuss other possible applications.

First, given a diagnostic hypothesis, we typically have the opportunity to gather additional information about the state of the world before taking further actions. Value of information is a concept that captures this task (Howard 1966). Existing diagnostic systems often calculate myopic value of information and recommend a single best test to perform. However, it is possible that each single test has value of information less than its cost, but there exists some set of tests whose value of information is greater than their cost (Heckerman, Horvitz, & Middleton 1993). In this case, myopic value of information will incorrectly stop any information gathering because it believes no test is cost effective. Optimal non-myopic value of information has to consider all possible test combinations. Such an analysis is intractable unless only a small number of tests is under consideration. We can apply eMAP in this setting to select one or multiple subsets of the tests and consider different test combinations only within these test sets.

Second, eMAP can be used to perform model evaluation. After we build a model, it is important to evaluate the performance of the model before applying it to real problems. One approach is to use real cases to evaluate the model. However, real cases are typically difficult and expensive to obtain. The few cases that we manage to get are also not enough to evaluate the model thoroughly. eMAP can be applied in this setting to simulate faults and generate test cases as in (Lu & Przytula 2007). For this purpose, we can use eMAP to simulate the most probable observations given faults as our test cases. We can also use eMAP to find the most probable explanations for simulated test cases.

Conclusion and Discussion

In this paper, we propose a novel approach for finding the best explanations for new observations in Bayesian networks. Based on our analysis on an example model, eMAP demonstrates the capability to automatically identify the most relevant variables in explaining the evidence. eMAP using Bayes factor can factor in both prior and posterior probabilities properly in ranking the potential explanations and was able to distinguish between explanations that likelihood cannot distinguish. eMAP has huge potential to many diagnosis problems, such as sequential diagnosis, fault simulation, model evaluation, and value of information. In our future work, we plan to design and implement efficient exact and approximate algorithms for eMAP, and, based on that, develop methodologies for multiple-fault diagnosis.

Acknowledgements

All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory and available at `http://genie.sis.pitt.edu`.

References

Akaike, H. 1974. A new look at the statistical model identification. *IEEE Transactions on Automatic Control* 19(6):716723.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in bayesian networks. In *UAI*, 115–123.

Chajewska, U., and Halpern, J. Y. 1997. Defining explanation in probabilistic systems. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, 62–71. San Francisco, CA: Morgan Kaufmann Publishers.

de Campos, L. M.; Gamez, J. A.; and Moral, S. 2001. Simplifying explanations in Bayesian belief networks. *International Journal of Uncertainty, Fuzziness Knowledge-Based Systems* 9(4):461–489.

Fitelson, B. 2001. *Studies in Bayesian confirmation theory*. Ph.D. Dissertation, University of Wisconsin, Madison, Philosophy Department.

Gärdenfors, P. 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press.

Good, I. 1985. Weight of evidence: A brief survey. *Bayesian Statistics* 2:249–270.

Heckerman, D.; Horvitz, E.; and Middleton, B. 1993. An approximate nonmyopic computation for value of information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(3):292–298.

Henrion, M., and Druzdzel, M. J. 1991. Qualitative propagation and scenario-based schemes for explaining probabilistic reasoning. In Bonissone, P.; Henrion, M.; Kanal, L.; and Lemmer, J., eds., *Uncertainty in Artificial Intelligence 6*. New York, N. Y.: Elsevier Science Publishing Company, Inc. 17–32.

Howard, R. A. 1966. Information value theory. *IEEE Transactions on Systems Science and Cybernetics* SSC2(1):22–26.

Jeffreys, H. 1961. *Theory of Probability*. Oxford University Press.

Lu, T.-C., and Przytula, K. W. 2006. Focusing strategies for multiple fault diagnosis. In *Proceedings of the 19th International FLAIRS Conference (FLAIRS-06)*, 842–847.

Lu, T.-C., and Przytula, K. W. 2007. System diagnosability analysis using p-slop MAP. In *Proceeding of the 20th International FLAIRS Conference (FLAIRS-07)*, To appear.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Poole, D., and Provan, G. M. 1991. What is the most likely diagnosis? In Bonissone, P.; Henrion, M.; Kanal, L.; and Lemmer, J., eds., *Uncertainty in Artificial Intelligence 6*. New York, N. Y.: Elsevier Science Publishing Company, Inc. 89–105.

Shimony, S. E. 1993. The role of relevance in explanation I: Irrelevance as statistical independence. *International Journal of Approximate Reasoning* 8(4):281–324.

Author Index