Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08)

September 22–24, 2008

Blue Mountains, NSW, Australia

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors
Foreword

The International Workshop on Principles of Diagnosis is an annual event that started in 1989, originating 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 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 retained the enlarged program committee introduced with DX–07 and have again received a large number of submissions from 14 countries. Each paper was thoroughly peer reviewed by three reviewers. We accepted 26 full papers and 19 posters.

We wish to thank all the authors of submitted papers, the program committee members for the time and effort spent, the panel members and the invited speaker for their participation.

We thank the team involved in local organisation: Lachlan Blackhall and Priscilla Kan John at NICTA/ANU, and Judy Pollock at the Advanced Computing Research Centre, University of South Australia. The workshop would not have been possible without their help.

We also wish to thank our sponsors: the University of South Australia (UniSA), the Australian National University (ANU), the National Aeronautics and Space Administration (NASA Ames), Australia’s Information and Communications Technology Centre of Excellence (NICTA), and the Collaborative Research Centre for Integrated Engineering Asset Management (CIEAM CRC).

Alban Grastien, Markus Stumptner, and Wolfgang Mayer  
September 2008
Workshop Organization

Program Co-Chairs
Alban Grastien  NICTA and Australian National University, Canberra, Australia
Markus Stumptner  University of South Australia, Adelaide, Australia
Wolfgang Mayer  University of South Australia, Adelaide, Australia

International Program Committee
Sherif Abdelwahed  Vanderbilt University, USA
Carlos J Alonso  University of Valladolid, Spain
Gautam Biswas  Vanderbilt University, USA
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
Eric Fabre  INRIA/IRISA, France
Alban Grastien  NICTA and Australian National University, Australia
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
Xenofon Koutsoukos  Vanderbilt University, USA
Stephane Lafortune  University of Michigan, USA
Jan Lunze  Ruhr-Universität Bochum, Germany
Wolfgang Mayer  University of South Australia, Australia
Pieter Mosterman  The MathWorks, Inc., USA
Sriram Narasimhan  NASA Ames, USA
Mattias Nyberg  Scania CV Sodertalje, Sweden
Yannick Pencolé  CNRS/LAAS, France
Gregory Provan  Cork College, Ireland
Belarmino Pulido  University of Valladolid, Spain
Liu Qiao  Toyota Technical Center, USA
John Sheppard  Johns Hopkins University, USA
Marcel Staroswiecki  Lille University, France
Peter Struss  Technical University of Munich, Germany
Markus Stumptner  University of South Australia, Australia
Timothy Wilmring  Boeing PhantomWorks, USA
Franz Wotawa  Graz University of Technology, Austria
Mariana Zanella  University of Brescia, Italy
Additional Reviewers

Jinbo Huang \hspace{1cm} NICTA and Australian National University, Australia
Anika Schumann \hspace{1cm} Technical University of Munich, Germany
Louise Trévé-Massuyès \hspace{1cm} University of Toulouse, France

Workshop Organizing Committee

Lachlan Blackhall \hspace{1cm} Australian National University, Australia
Alban Grastien \hspace{1cm} NICTA and Australian National University, Australia
Priscilla Kan \hspace{1cm} NICTA and Australian National University, Australia
Wolfgang Mayer \hspace{1cm} University of South Australia, Australia
Markus Stumptner \hspace{1cm} University of South Australia, Australia
Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
Current Fault Management Trends in NASA’s Planetary Spacecraft

Lorraine Fesq and Michel Ingham
Jet Propulsion Laboratory, California Institute of Technology

Abstract
Fault management for today’s space missions is a complex problem, going well beyond the typical safing requirements of simpler missions. Recent missions have experienced technical issues late in the project lifecycle, associated with the development and test of fault management capabilities, resulting in both schedule delays and cost overruns. Symptoms seem to become exaggerated in the context of deep space and planetary missions, most likely due to the need for increased autonomy and the limited communication opportunities with Earth-bound operators. These issues are expected to be further exacerbated as the spacecraft envisioned for future missions become more capable and complex. In recognition of the importance of addressing this problem, the Discovery and New Frontiers Program Office hosted a Fault Management Workshop on behalf of NASA’s Science Mission Directorate, Planetary Science Division, to bring together experts in fault management from across NASA, DoD, industry, and academia. The scope of the workshop was focused on deep space and planetary robotic missions, with full recognition of the relevance of, and subsequent benefit to, Earth-orbiting missions. The following three topics, in particular, were targeted in the workshop breakout sessions:

1. Fault Management Architectures
2. Fault Management Verification and Validation

The key product of this three-day workshop is a NASA report documenting lessons learned from previous missions, recommended best practices, and future opportunities for investments in the fault management domain. An emerging realization from the workshop is the high cost and risk of proceeding with business as usual in the area of fault management engineering.

A few of the primary findings include:

- Spacecraft fault management currently is engineered in a fairly ad hoc manner, both from a development process and from a software architecture point of view. There exist opportunities to (i) establish principles to guide the design and implementation of fault management systems, and (ii) identify ways to improve communications within the field such as establishing a standardized vocabulary and adopting a formal representation language to express the designs.
- The implementation of fault management software architectures is very similar across the government and industry organizations that develop planetary spacecraft. Most fault management systems are fundamentally designed as monitor-and-response systems, running independently from the nominal execution mechanisms.
- Planetary missions would benefit from earlier consideration of fault management in the spacecraft engineering lifecycle. For example, capabilities for system behavioral modeling and complexity analysis could be brought to bear during early system design. This may require development of new tools for modeling and analysis, or the application of existing tools, and their integration into the mission formulation and pre-formulation design environment (e.g., performance and costing models used for early system trades).
- Missions are not yet leveraging technologies at their disposal, such as those developed by the DX community. This is in part a reflection of (i) the risk-averse nature of the spacecraft engineering community, (ii) the challenge posed by the technology maturation gap the scarcity of funding that allows technologies to mature through the mid-Technology Readiness Level range, and (iii) the challenges associated with effectively marketing technology (especially software technology) to missions and targeting their specific requirements.

This presentation will provide an overview of the findings from the workshop, summarize the recommended best practices, and highlight some opportunities for near-term and strategic investment of particular interest to the DX community.

Acknowledgments
The authors would like to thank Jim Adams, Deputy Director of the Planetary Science Division in the Science Mission Directorate at NASA Headquarters, for instigating the organization of the Fault Management Workshop, and for providing programmatic leadership and funding for this activity. We also acknowledge Paul Gilbert, Manager of the Discovery & New Frontiers Program Office, as programmatic...
host of the Workshop. Organization of the workshop was very much a cross-Agency team effort involving the members of our workshop steering committee: John McDougal (NASA Marshall Space Flight Center), Chris Jones (Caltech Jet Propulsion Lab), George Cancro (Johns Hopkins University Applied Physics Lab), Steven Scott (NASA Goddard Space Flight Center), and Raymond Whitley (NASA Goddard Space Flight Center). Our thanks also go out to David Watson (Johns Hopkins University Applied Physics Lab), Marilyn Newhouse (Computer Sciences Corporation) and Julie Wertz (Caltech Jet Propulsion Lab), for their commitment and hard work in planning and executing the workshop.
A Dynamic Modeling Approach to Software Multiple-Fault Localization

Rui Abreu and Peter Zoeteweij and Arjan J.C. van Gemund

Embedded Software Lab
Delft University of Technology
The Netherlands
{r.f.abreu,p.zoeteweij,a.j.c.vangemund}@tudelft.nl

Abstract

Current model-based approaches to software debugging use static program analysis to derive a model of the program. In contrast, in the software engineering domain diagnosis approaches are based on analyzing dynamic execution behavior. We present a model-based approach where the program model is derived from dynamic execution behavior, and evaluate its diagnostic performance for both synthetic programs and the Siemens software benchmark, extended by us to accommodate multiple faults. We show that our approach outperforms other model-based software debugging techniques, which is partly due to the use of De Kleer’s intermittency model to account for the variability of software component behavior.

1 Introduction

Automatic software fault localization techniques aid developers to pinpoint the root cause of failures, thereby reducing the debugging effort. Two major approaches can be distinguished, (1) the spectrum-based fault localization (SFL) approach, a statistical approach that correlates dynamic software component activity (i.e., execution traces) with program failures [1; 12; 13], and (2) the model-based diagnosis (MBD) approach, which deduces component failure through logic reasoning over a static model of the program [4; 5; 6; 7; 8; 9; 15; 16; 20].

Because of its low computational complexity and absence of modeling requirements, SFL has gained large popularity in the software engineering community. Although inherently not restricted to single faults, in most cases these statistical techniques are applied and evaluated in a single-fault context, such as the Siemens benchmark set [10], which is seeded with only 1 fault per program (version). In practice, however, the defect density of even small programs typically amounts to multiple faults. Although the root cause of a particular program failure need not constitute multiple faults that are acting simultaneously, many failures will be caused by different faults. Hence, the problem of multiple-fault localization (diagnosis) deserves detailed study.

Unlike SFL, MBD inherently considers multiple faults. However, the logic models of software systems that are used in the diagnostic inference are typically based on static program analysis. Consequently, they do not consider dynamic execution behavior, such as (data-dependent) conditional control flow, which, in contrast, forms the essence of the SFL approach. Aimed to combine the best of both worlds, in this paper we present an approach that exploits the dynamic, execution trace-based observation approach from SFL, to derive models and observations as input to MBD to produce multiple-fault diagnoses.

To illustrate the potential advantages of a multiple-fault approach, consider a triple-fault program with faulty components \(c_1, c_2,\) and \(c_3\). Whereas (ideally) a single-fault approach such as SFL would produce multiple single-fault diagnoses like \(\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \ldots\}\) (component indices, ordered in terms of statistical similarity), a multiple-fault approach would simply produce one single multiple-fault diagnosis \(\{\{1, 2, 3\}\}\). This single diagnosis unambiguously reveals the actual triple fault, which, additionally, measures the potential for debugging parallelism [11], whereas in the former case it is not obvious how many faults are actually present.

This paper makes the following contributions:

- We present our multiple-fault diagnosis method which combines a dynamic modeling and observation approach known from SFL with a diagnostic reasoning approach from MBD.
- We evaluate our approach using the Siemens set benchmark, extended by us to accommodate multiple faults.
- We evaluate the merit of two specific strategies for updating the probabilities of diagnosis candidates, based on De Kleer’s intermittent fault model [4], to account for the fact that faulty (software) components very often exhibit nominal behavior.
- We compare our approach to related reasoning approaches (AIM [15], \(\Delta\)-slicing [8], and explain [8]) for the Siemens set program tcas (their common benchmark).

Our experiments show that strategies that exploit (intermittency) information to exonerate components involved in
passed runs outperform those that do not include such information. Furthermore, experiments using the \texttt{tcas} program show that our approach allows that more bugs can be solved if limited debugging time is available.

2 Preliminaries

In this section we introduce the concepts and definitions used throughout this paper.

2.1 Basic Definitions

Definition 1 A diagnostic system \( DS = (SD, COMPS, OBS) \), where \( SD \) is a propositional theory describing the behavior of the system, \( COMPS = \{c_1, \ldots, c_M\} \) is a set of components in \( SD \), and \( OBS \) is a set of observable variables in \( SD \).

With each component \( c_m \in COMPS \) we associate a health variable \( h_m \) which denotes component health. The health states of a component are healthy (true) and faulty (false).

Definition 2 An h-literal is \( h_m \) or \( \neg h_m \) for \( c_m \in COMPS \).

Definition 3 An h-clause is a disjunction of h-literals containing no complementary pair of h-literals.

Definition 4 A conflict of \( (SD, COMPS, OBS) \) is an h-clause of negative h-literals entailed by \( SD \cup OBS \).

Definition 5 Let \( S_N \) and \( S_P \) be two disjoint sets of components indices, faulty and healthy, respectively, such that \( COMPS = \{c_m \mid m \in S_N \cup S_P\} \) and \( S_N \cap S_P = \emptyset \). We define \( d(S_N, S_P) \) to be the conjunction:

\[
\left( \bigwedge_{m \in S_N} \neg h_m \right) \land \left( \bigwedge_{m \in S_P} h_m \right)
\]

A diagnosis candidate is a sentence describing one possible state of the system, where this state is an assignment of the status healthy or not healthy to each system component.

Definition 6 A diagnosis candidate for \( (SD, COMPS, OBS) \), given an observation term \( obs \) over variables in \( OBS \), is \( d(S_N, S_P) \) such that

\[
SD \land obs \land d(S_N, S_P) \not\models \bot
\]

In the remainder we refer to \( d(S_N, S_P) \) simply as \( d \), which we identify with the set \( S_N \) of indices of the negative literals.

Definition 7 A diagnosis \( D = \{d_1, \ldots, d_K\} \) is an ordered set of all \( K \) diagnosis candidates, for which \( SD \land obs \land d_k \not\models \bot \).

2.2 Model-based Diagnosis

In this section we describe the principles underlying model-based software diagnosis as far as relevant to this paper.

Consider the simple program function in Figure 1, which is supposed to be composed of three inverting statements (with a fault in statement 3), resembling a circuit with three logical inverters\(^1\). The function takes one input \( x \), and returns

\[
(y_1, y_2) = 3\text{inv} (\text{bool} x) \{
1. \ w = \neg x \\
2. \ y_1 = !w; \\
3. \ y_2 = w; \ // \text{fault: ! missing return (y_1, y_2); }
\}
\]

Figure 1: A defective function

two outputs \( y_1, y_2 \). A weak model of each inverter statement, which only specifies nominal (required) behavior, is given by the proposition

\[ h \Rightarrow y = \neg x \]

Given the data dependencies of the program, the interconnection topology of the three inverting components is easily obtained, yielding the (combined) program model

\[
h_1 \Rightarrow w = \neg x \]
\[
h_2 \Rightarrow y_1 = \neg w \]
\[
h_3 \Rightarrow y_2 = \neg w
\]

Computing Diagnoses

Consider the observation \( obs = ((x, y_1, y_2) = (1, 1, 0)) \). From the model, it follows

\[
h_1 \Rightarrow \neg w \\
h_2 \Rightarrow \neg w \\
h_3 \Rightarrow w
\]

which equals

\[
(\neg h_1 \vee \neg w) \land (\neg h_2 \vee \neg w) \land (\neg h_3 \vee w)
\]

Resolution yields the following conjunction of conflicts

\[
(\neg h_1 \vee \neg h_3) \land (\neg h_2 \vee \neg h_3)
\]

meaning that (1) at least \( c_1 \) or \( c_3 \) is at fault, and (2) at least \( c_2 \) or \( c_3 \) is at fault. The minimal diagnoses are given by the minimal hitting set \([18]\), yielding

\[
\neg h_3 \vee (\neg h_1 \land \neg h_2)
\]

Thus either \( c_3 \) is at fault (single fault), or \( c_1 \) and \( c_2 \) are at fault (double fault), as well as a number of other double-faults \((\neg h_2 \lor h_3, \neg h_1 \lor h_3)\), and a triple fault \((\neg h_1 \lor h_1 \lor h_3)\), which, however, are subsumed by the previous two minimal diagnoses. Consequently, \( D = \{\{3\}, \{1, 2\}\} \).

Ranking Diagnoses

The fact that models do not always specify all possible behavior (e.g., weak models), and that usually only limited observations are available typically leads to diagnoses with many solutions. However, not all solutions are equally probable, allowing them to be ranked in order of probability of being the actual fault state.

Let \( Pr(\{j\}) \) denote the a priori probability that a component \( c_j \) is at fault. Although this value is typically component-specific, in the above inverter example we assume \( Pr(\{j\}) = p \) (where we arbitrarily set \( p = 0.01 \)). Assuming components fail independently, and in absence of any observation,

\(^1\)Note that in this approach we assume the presence of a correct model of the components (in contrast to e.g. [14])

the prior probability a particular diagnosis \(d_k\) is correct is given by

\[
\Pr(d_k) = \prod_{j \in S_n} \Pr(\{j\}) \cdot \prod_{j \in S_p} (1 - \Pr(\{j\}))
\]

In order to compute the posterior probability given an observation we present our dynamic, observation-based diagnosis approach (expressing fault coverage, also known as the “goodness” parameter \(g\) [4]).

Note that at least one faulty component has to be involved in the computation of a failed run. From \(O\) it is also possible to derive the probability \(r\) that a component is actually executed in a run (expressing code coverage), and the probability \(g\) that a faulty component is actually exhibiting good behavior (expressing fault coverage, also known as the “goodness” parameter \(g\) [4]).

### 3 Computing Diagnoses

Unlike the MBD approach mentioned earlier, which statically deduces information from the program source, \(O\) is the only, dynamic source of information, from which both a model, and the input-output observations are derived. Apart from exploiting dynamic information, this approach only requires a generic component model, avoiding the need for detailed functional modeling or relying, e.g., on invariants or pragmas for model information. Note, however, that this default model can easily be extended when more detailed information is available.

Abstracting from particular component behavior, each component \(c_j\) is modeled by the weak model

\[
h_j \models (x_j \Rightarrow y_j)
\]

where \(h_j\) models the health state of \(c_j\) and \(x_j, y_j\) model its input and output variable value correctness (i.e., we abstract from actual variable values, in contrast to the earlier example). This weak model implies that a healthy component \(c_j\) translates a correct input \(x_j\) to a correct output \(y_j\). However, a faulty component or input may lead to an erroneous output.

As each row in \(O\) specifies which components were involved, we interpret a row as a “run-time” model of the program as far as it was considered in that particular run. Consequently, \(O\) is interpreted as a sequence of typically different models of the program, each with its particular observation of input/output correctness. The overall diagnosis can be viewed as a sequential diagnosis approach that incrementally takes into account new program state (and pass/fail) evidence with increasing \(N\). A single row \(O_{n,*}\) corresponds to the (sub)model

\[
h_m \models (x_m \Rightarrow y_m), \text{ for } m \in T_n \\
x_{t_i} = y_{t_i-1}, \text{ for } i \geq 2 \\
x_{t_1} = \text{true} \\
y_w = \neg e_n
\]

where \(T_n = \{m \in \{1, \ldots, M\} \mid o_{nm} = 1\}\) denotes the well-ordered set of component indices involved in computation \(n\), \(t_i\) denotes the \(i^{th}\) element in this ordering, (i.e., for \(i \leq j, t_i \leq t_j\), \(t_i\) denotes its last element. The resulting component chain logically reduces to

\[
\bigwedge_{m \in T_n} h_m \Rightarrow \neg e_n
\]

For example, consider the row \((M = 5)\)

<table>
<thead>
<tr>
<th>(c_1)</th>
<th>(c_2)</th>
<th>(c_3)</th>
<th>(c_4)</th>
<th>(c_5)</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

This corresponds to a model where components \(c_1, c_4\) are involved. As the order of the component invocation is not given
(and with respect to our above weak component model is irrelevant), we derive the model
\[ h_1 \Rightarrow (x_1 \Rightarrow y_1) \]
\[ h_4 \Rightarrow (x_4 \Rightarrow y_4) \]
\[ x_4 = y_1 \]
\[ x_1 = \text{true} \]
\[ y_4 = \neg e_n \]

In this chain the first component \( c_1 \) is assumed to have correct input \((x_1 = \text{true}, \text{typical of a proper test})\), its output feeds to the input of the next component \( c_4 \) \((x_4 = y_1)\), whose output is measured in terms of \( e_n \) \((y_4 = \neg e_n)\). This chain logically reduces to
\[ h_1 \land h_4 \Rightarrow \text{false} \]

If this were a passing computation \((h_1 \land h_4 \Rightarrow \text{true})\) we could not infer anything (apart from the exoneration when it comes to probabilistically rank the diagnosis candidates as explained in next section). However, as this run failed this yields
\[ \neg h_1 \lor \neg h_4 \]

which, in fact, is a conflict. In summary, each failing run in \( O \) generates a conflict according to
\[ \bigvee_{m \in T_m} \neg h_m \]

As in the former MBD approach, the conflicts are then subject to a hitting set algorithm that generates the diagnostic candidates.

To illustrate this concept, again consider the example program. For the purpose of the spectral approach we assume the program to be run two times where the first time we consider the correctness of \( y_1 \) and the second time \( y_2 \). This yields the observation matrix \( O \) below

\[
\begin{array}{c|c|c|c|c|c}
  c_1 & c_2 & c_3 & e & obs_1 & obs_2 \\
  \hline
  1 & 1 & 0 & 0 & obs_1 \\
  1 & 0 & 1 & 1 & obs_2 \\
\end{array}
\]

From \( obs_2 \), it follows
\[ \neg h_1 \lor \neg h_3 \]

which equals the first conflict from the earlier MBD approach, and the diagnosis trivially comprises the two single faults \( \{1\} (\neg h_1) \) and \( \{3\} (\neg h_3) \). Compared to the earlier MBD approach, the second conflict \((\neg h_2 \lor \neg h_3) \) is missing due to the fact that no additional knowledge is available on component behavior and component interconnection. Although this would suggest that the dynamic approach yields lower diagnostic performance than the earlier MBD approach, note that the example program is ideally suited to static analysis, whereas real programs feature extensive control flow, rendering the static approach extremely difficult. However, if, for some reason, we were able to capture the second conflict in terms of an execution trace according to

\[
\begin{array}{c|c|c|c|c|c}
  c_1 & c_2 & c_3 & e & obs_3 \\
  \hline
  0 & 0 & 1 & 1 & obs_3 \\
\end{array}
\]

then our observation-based approach would yield exactly the same set of minimal diagnoses.

Note that, e.g., unlike dynamic slicing [21] and constraint-based models [17], we do not exploit actual data dependencies between components but execution patterns.

### 3.3 Ranking Diagnoses

Similar to the incremental compilation of conflicts per run we compute the posterior probability for each candidate based on the pass/fail observation \( obs \) for each sequential run using Bayes’ rule as described in Section 2.2. In the following we will distinguish between three \( \varepsilon \) policies. The first policy, denoted \( \varepsilon^{(0)} \) is similar to the classical MBD policy, and is defined as follows
\[
\varepsilon^{(0)} = \begin{cases} 
  E_P & \text{if run passed} \\
  \frac{E_P}{E_P+E_F} & \text{if run failed}
\end{cases}
\]

where \( E_P = 2^M \) and \( E_F = (2^l - 1) \cdot 2^{M-l} \) are the number of passed and failed observations that can be explained by diagnosis \( d_k \), respectively, and \( l = |d_k| \) is the number of faulty components in the diagnosis. Note that this policy is slightly different from the one in Section 2.2, as the lack of component interconnection information allows more diagnoses (component combinations) as likely explanations for pass/fail outcomes.

A disadvantage of this classical policy is that passed runs, apart from making single faults more probable than multiple faults, do not help much in pinpointing the fault location. This has to do with the fact that all diagnoses are possible when a run passes due to the weak fault model (the \( 2^M \) term in Eq. 1). In addition, there is no way to distinguish between diagnoses with the same cardinality, because the terms are merely a function of the cardinality of the diagnosis candidate.

An approach to account for the fact that, similar to statistical approaches for fault localization, components involved in passed computations should to some extent be exonerated, is by extending the component model with an intermittent failure model, as introduced by De Kleer [4]. As in software components it is quite usual that a faulty component exhibits correct behavior, we include statistical information on the probability that a faulty component \( e \) exhibits correct behavior. Let \( g(d_k) \) denote the aforementioned (“goodness”) probability that faulty components in \( d_k \) are exhibiting good behavior. In the following we distinguish between two different policies, which we refer to as \( \varepsilon^{(1)} \), and \( \varepsilon^{(2)} \), which are defined as follows
\[
\varepsilon^{(1)} = \begin{cases} 
  g(d_k) & \text{if run passed} \\
  1 - g(d_k) & \text{if run failed}
\end{cases}
\]

and
\[
\varepsilon^{(2)} = \begin{cases} 
  g(d_k)^t & \text{if run passed} \\
  1 - g(d_k)^t & \text{if run failed}
\end{cases}
\]

where \( t \) is the number of faulty components according to \( d_k \) involved in the run \( i \)

\[ t = \prod_{j \in d_k} |a_{ij} = 1| \]

We propose policy \( \varepsilon^{(2)} \) as a variant of \( \varepsilon^{(1)} \), which is due to De Kleer [4]. It approximates the probability \( \sum_{j \in d_k} g_j \) that the components in \( d_k \) all exhibit good behavior by \( g(d_k)^t \), as-
summing that all components of \( d_k \) have equal goodness probabilities. In both strategies we use
\[
g(d_k) = \frac{\sum_{i=1..N} \left[ (\bigvee_{j \in d_k} a_{ij} = 1) \land e_i = 0 \right]}{\sum_{i=1..N} \left[ \bigvee_{j \in d_k} a_{ij} = 1 \right]}
\]
where \([\cdot]\) is Iverson’s operator (\([\text{true}] = 1\), \([\text{false}] = 0\)). In the above two policies \( g(\cdot) \) is assumed to be different from 0. Otherwise a strong exoneration factor is included, as a passed run would set \( Pr(d_k) = 0 \), for all diagnosis candidates. Furthermore, one can think of additional policies, such as combining either \( \varepsilon(1) \) or \( \varepsilon(2) \) with \( \varepsilon(0) \). Note, however, that in this case the variable term in \( \varepsilon(0) \) is a function of the diagnosis’ cardinality only. As the prior probabilities \( p \) already differentiate diagnoses with different cardinalities, such a combined policy would not produce a different ranking than those obtained by either \( \varepsilon(1) \) or \( \varepsilon(2) \).

\[
\begin{array}{c|ccc}
\varepsilon(0) & \varepsilon(1) & \varepsilon(2) \\
Pr(1) & 0.5 & 0.2 & 0.2 \\
Pr(3) & 0.5 & 0.8 & 0.8 \\
\end{array}
\]

(a) After \( obs_1 \) and \( obs_2 \)

\[
\begin{array}{c|ccc}
\varepsilon(0) & \varepsilon(1) & \varepsilon(2) \\
Pr(3) & 0.98 & 0.99 & 0.999 \\
Pr(1,2) & 0.02 & 0.01 & 0.001 \\
\end{array}
\]

(b) After \( obs_1 \), \( obs_2 \), and \( obs_3 \)

Figure 2: Probabilities updates

Returning to the example of Section 2.2, Figure 2 lists the probabilities resulting from the various \( \varepsilon \) policies for the diagnoses obtained after \( obs_1 \) and \( obs_2 \) only (Figure 2(a)) and after \( obs_3 \) (Figure 2(b)). In the first case, the classical policy cannot distinguish between \( c_1 \) and \( c_3 \) while the \( g \) policies exploit the additional information provided by the exonerating observation \( obs_1 \). When \( obs_3 \) is included \( c_1 \) is no longer a valid diagnosis by itself, and is eliminated from the (hitting) set of valid diagnosis candidates. Hence, all policies favor \( c_3 \) as most likely candidate, due to (1) the lower prior probability of the double fault (\( \varepsilon(0,1,2) \)) and (2) the exoneration by passed runs (\( \varepsilon(1,2) \)).

4 Theoretical Evaluation

In order to gain understanding of the effects of the various parameters on the diagnostic performance of the different policies, in this section we use a simple, probabilistic model of program behavior that is directly based on \( N, M, r, \) and \( g \). Without loss of generality we model the first \( C \) of the \( M \) components to be at fault (\( C \) for fault cardinality). For each run each component has probability \( r = 0.6 \) (which go in accordance with the value measured for our software benchmark set of faults, see next section) to be involved in that run. If a selected component is faulty, the probability of exhibiting nominal (“good”) behavior equals \( g \). When either of the \( C \) components fails, the run will fail. We study the performance of the \( \varepsilon \) policies defined previously for observation matrices that are randomly generated according to the above model.

4.1 Performance Metrics

Before evaluating the results, we first present our performance metric. Fault localization techniques aim at helping developers in finding bugs quickly, and a metric to evaluate such techniques is to measure the amount of code a developer would have to inspect before (but not including) finding the fault cause, wasted effort \( W \). It is defined as the number of inspected components divided by the total number of components (\( M \)). In our computation of \( W \) we assume that after each inspection, the test set is rerun, possibly leading to a new ranking (without the most recently removed fault). For example, suppose a triple-fault program (\( M = 6 \), and \( c_1, c_2, \) and \( c_3 \) faulty) for which the following diagnosis \( D = \{1, 2, 6\}, \{3, 4, 5\}\) is obtained. This diagnosis induces a wasted effort of \( W = 33% \) as \( c_6 \) in the first candidate is inspected in vain, as well as, on average two out of three inspections in the second candidate (in this example we assumed that rerunning the test set did not change the second candidate \( \{3, 4, 5\}\). For example, had the two components in the second diagnosis candidate been inspected, then \( W = 50\% \).

In contrast to related work, we measure \( W \) instead of effort \( [1; 19] \) so that the performance metric’s scale is independent of the number of faults in the program. Another reason not to adopt the aforementioned score metric is that in our synthetic model we do not have program dependence graph information. For the example given above, the effort as defined in \([1; 19]\) would be 100%.

4.2 Diagnosis Optimality

As mentioned in the Introduction, under ideal circumstances our multiple-fault approach produces one single multiple-fault diagnosis \( \{1, \ldots, C\} \). This optimal result (shown in [2]) is obtained (for programs where each component has an independent, non-zero probability of being involved in any run) when \( N \to \infty \). This can be seen through the following argument. Consider a \( C \)-fault program. While for small \( N \) the minimal hitting set will still contain many members (components) other than the \( C \) faulty components, by increasing \( N \) the probability that a non-faulty component will still be included steadily decreases. Let \( f \) denote the probability of a run failing (derived as function of \( r, g, C \) in [2]). For the hitting set analysis only failing runs matter. For those \( N_F = f \cdot N \) failing runs the \( C \)-fault candidate is by definition within the set of candidates that “survive” those runs (whose chain is still unbroken). However, the probability that other components can be involved in a candidate is less than unity, which forms the basis of those candidates’ eventual elimination. In the following experiments this dependency of \( W \) on \( N \) is further studied.

4.3 Experimental Results

In this section we experimentally study the diagnostic performance of the different policies. For that purpose, a synthetic observation matrix generator is implemented, which takes into account \( N, g, \) and \( C \). Although we verified the influence of these parameters, in the following \( M \) is fixed to 20 and \( r \) to 0.6 as they do not change our conclusions.
Figure 3 contains plots of $W$ versus $N$ for $C = 1$, $C = 2$ and $C = 5$. Each measurement represents an average over 1,000 sample matrices. The plots show that $W$ for $N = 1$ is similar to $r$, which follows from the fact that there are on average $(M - C) \cdot r$ components which would have to be inspected in vain. For sufficiently large $N$ all policies produce an optimal diagnosis, see previous section. For small number of runs $N$, given the fact that it does not distinguish between diagnosis with the same fault cardinality (see Section 3.3), $\varepsilon^{(0)}$ is the worst performing policy. For $C \geq 2$, $\varepsilon^{(2)}$ outperforms $\varepsilon^{(1)}$, suggesting that information on the number of components involved a run should be included ($t$ in $\varepsilon^{(2)}$).

For $C = 1$, as $t = 1$ the performance of $\varepsilon^{(1)}$ equals the one of $\varepsilon^{(2)}$.

Furthermore, from the plots we verify that the higher $C$ the more runs $N$ are needed to attain optimal diagnostic performance. As an example, for $g = 0.1$, $r = 0.4$, and $C = 1, 13$ runs would be enough to yield a perfect diagnosis, whereas for $C = 5$, 250 runs would be needed. We have determined the value of $N$ ($N_F$) for which our $C$-cardinality remain as the only candidate, i.e., a perfect multiple-fault diagnosis $\{1, \ldots, C\}$. Table 1 shows the values of $N$ ($N_F$) where optimality is reached for different values of $C$ and $g$. Apart from a scaling due to $g$ one can clearly see the exponential impact of $C$ on $N_F$ and $N$ (shown in [2]).

<table>
<thead>
<tr>
<th>$g$</th>
<th>$r = 0.1$</th>
<th>$r = 0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>$N_F$</td>
<td>120</td>
<td>1000</td>
</tr>
<tr>
<td>$N$</td>
<td>300</td>
<td>1500</td>
</tr>
</tbody>
</table>

Table 1: Optimal $N^*$ for perfect diagnosis ($r = 0.6$)

5 Experimental Evaluation

In this section we assess the diagnostic capabilities of the dynamic modeling approach for real programs. For this purpose, we use the well-known Siemens set [10], which contains 132 faulty versions of 7 C programs with extensive test suites. Table 2 summarizes the characteristics of the Siemens set, where $M$ corresponds to the number of lines of code (components in this context).

For our experiments, we have extended the Siemens set with program versions in which we can activate arbitrary combinations of faults. For this purpose, we limit ourselves to a selection of 102 out of the 132 faults, based on criteria such as faults being attributable to a single line of code, to enable unambiguous evaluation. The observation matrices are obtained using the GNU gcov\(^2\) profiling tool.

Using this extended Siemens set, we evaluate our dynamic modeling approach in two ways: first, in Section 5.1, we measure its diagnostic performance on single and multiple-fault programs for the three $\varepsilon$ strategies outlined in Section 2.2. Next, in Section 5.2 we compare this performance against other diagnosis techniques. Here we use single-fault versions of the tcas program, which is the common program used in literature to evaluate these other techniques.

5.1 Results

Table 3 lists the wasted effort $W$, as defined in Section 4.1, incurred by the dynamic modeling approach and strategies $\varepsilon^{(0)}$, $\varepsilon^{(1)}$, and $\varepsilon^{(2)}$ for debugging single, double, and multiple-fault programs. Like in Section 4.3, we aimed at $C = 5$ for the multiple fault-cases, but for print_tokens insufficient faults are available, and for print_tokens2 and replace our current implementation of the hitting set algorithm practically prevents analyzing combinations of more

\(^2\)http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

---

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22-24, 2008, Blue Mountains, NSW, Australia.
than four and three faults, respectively.\textsuperscript{3} The hitting set computation is aborted after all diagnosis candidates with cardinality $C'$ have been generated. To simulate a more or less realistic debugging scenario, where the actual number of faults is unknown, we set $C' = \max(C, 3)$. All measurements except for the four-fault version of print\_tokens are averages over 100 versions, or over the maximum number of combination available, where we verified that all faults are active in at least one failed run.

Similar to what we observed in Section 4.3, in most cases $\varepsilon(1,2)$ lead to significantly better diagnoses than $\varepsilon(0)$ because they exonerate components that are involved in passed runs. The (minor) differences between the results for $\varepsilon(1)$ and $\varepsilon(2)$ at $C = 1$ are due to the different ways in which these strategies handle the diagnosis candidates of cardinality 2 and 3 that result from setting $C' = 3$.

Contrary to the results in Section 4.3, the improvement of $\varepsilon(2)$ over $\varepsilon(1)$ is marginal at best. We expect that this can be explained by using $g(d_k)$\textsuperscript{5} to approximate the product of the goodness parameters of the individual components in a diagnosis $d_k$, as explained in Section 3.3. In the context of strategy $\varepsilon(2)$, this entails using different goodness parameters for the same component as it occurs in different diagnoses, converging to the fraction of all runs that have passed as the diagnosis cardinality increases. The influence of these effects, and the unexpected superior performance of $\varepsilon(0)$ in the particular cases of schedule\_2 for $C = 2$ and $C = 5$, and tcas for $C = 5$ require further investigation.

### 5.2 Comparison

In the following we compare the diagnostic performance of our approach with AIM, nearest neighbor (NN), explain, and $\Delta$-slicing techniques (see Section 6 for a discussion). For compatibility with results reported for those techniques, we will use the effort, or score metric [1; 19] instead of wasted effort $W$ which amounts to the percentage of lines of code that need not be examined when the diagnosis results are used to guide the search for the fault. Note that the results reported for NN do not involve a full ranking of all statements, but are based on the distance between the fault and the diagnosis in the program dependence graph instead, which is a comparable measure [12].

Our current implementation of the dynamic modeling approach only supports C programs, while the AIM technique has mainly been evaluated for Java programs. The only C program that has been taken into account is tcas, which happens to be a common benchmark among the other techniques as well, so for this comparison we limit ourselves to that program. Furthermore, the other techniques have only been evaluated for single faults, so we set $C' = C = 1$, and therefore $\varepsilon(1) = \varepsilon(2)$.

Similar to the results in [15], in Table 4 we compare our approach with AIM and NN on tcas. As expected, $\varepsilon(0)$ is outperformed by all other techniques. AIM consistently outperforms NN. For an effort of less than 1%, $\varepsilon(1,2)$ outperform AIM, which yields the best results if 10% of the code is inspected. Both techniques find all faults by inspecting less than 20% of the code.

Table 5 compares the different policies used in our approach with AIM, explain, and $\Delta$-slicing for 5 versions of tcas, because these are the versions to which explain and $\Delta$-slicing could be applied to. From the table, we conclude that our approach when using $\varepsilon(1,2)$ consistently outperforms all other techniques, with $\varepsilon(0)$ being the worst performing technique.

### 6 Related Work

As mentioned in the introduction, automated debugging techniques can be distinguished into statistical and logic reasoning approaches that use program models.

In logic (model-based) reasoning approaches to automatic software debugging, the program model is typically generated from the source code. In [15] an overview of techniques based on automatically generated program models is given. They conclude that the models generated by means of abstract interpretation [14] (AIM approach) are the most accurate for debugging. Basically, a model of the (faulty) program is generated from the source code, e.g., using abstract interpretation, and the test cases specify the expected output. Differences between the program’s output and the expected one are used to compute components that when assumed to

\textsuperscript{3}A novel, statistics-directed improvement is currently under development. Preliminary results indicate orders of magnitude speedup.

---

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
behave differently explain the observed faulty behavior. Approaches based on model checkers include the explain [9], ∆-slicing [8], which are based on comparing execution traces of correct and failed runs. Although model-based diagnosis inherently considers multiple-faults, thus far the above software debugging approaches only consider single faults. Apart from the fact that our approach is multiple-fault, it also differs in the fact that we use program spectra as dynamic information on component activity, which allows us to exploit dynamic execution behavior, unlike static approaches. Furthermore, our approach does not rely on the approximations required by static techniques. In addition, similar to AIM, the approach presented in this paper does not require a formal specification of the program.

Statistical approaches are very attractive from complexity-point of view. Well-known examples are the Tarantula tool [12], the Nearest Neighbor technique [19], the Sober tool [13], and the Ochiai coefficient [1]. Although differing in the way they derive the statistical fault ranking, all techniques are based on measuring program spectra. All these approaches yields single fault explanations, whereas ours also includes multiple faults that explain all failures. This extra information may help to debug several failures in parallel, similar to [11] which uses clustering techniques to build sets of runs revealing the same failure.

7 Conclusions and Future Work
In this paper we present a dynamic modeling approach to software fault localization based on abstraction of program traces. The model, along with the set of traces for pass/fail executions is used to reason about observed failures. In contrast to most approaches to software fault diagnosis, which present diagnosis candidates as single explanations, our approach also contains multiple fault explanations in the diagnostic ranking (typical of model-based approaches).

We have evaluated the diagnostic performance of three Bayesian probability update policies, including De Kleer’s intermittency model and an extension proposed by us. Empirical results obtained from the widely-used Siemens set of programs, extended by us to accommodate multiple fault programs, show that policies that are able to exonerate components that are involved in passing runs clearly outperform the probability update scheme that is traditionally used in model-based diagnosis.

For future work, we plan to study whether the multiple-fault diagnosis candidates information can be used to efficiently engage several developers to repair the defect(s) in parallel, as well as incorporate our new algorithm to speedup the hitting set computation.

Acknowledgments
We extend our gratitude to Johan de Kleer for discussions which have influenced our multiple-fault reasoning approach.

References
The Challenge of Solving POMDPs for Control, Monitoring and Repair of Complex Systems

Emmanuel Benazera and Elodie Chanthery

1 LAAS-CNRS, Université de Toulouse, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4,
2 Université de Toulouse, INSA
benazer,elodie.chanthery@laas.fr

Abstract

A common view holds a clear separation between the act of diagnosing and that of planning/controlling. However, a framework exists, partially observable Markov decision problems (POMDPs), that is holistic for optimal decision and control in partially observable domains. In applications to systems with faults, its solution yields the best strategy that includes repair and observation actions. In such a decision-based context, it is unclear what a diagnostic is, and under what conditions it is useful or even necessary. This paper discusses the use of POMDPs to the control, monitoring and repair of systems with faults, insisting on their structure and showing how it can be used to improve the scalability of POMDP solving techniques for application to the present domain.

1 A well admitted problem in both scheduling and planning.

1 A well admitted problem in both scheduling and planning.

1.1 POMDPs as a holistic framework

A partially observable Markov decision process (POMDP) models control problems for which actions have stochastic effects and sensors provide imperfect and incomplete state information. This model has been widely adopted by the AI community as a framework for research in planning and control under uncertainty. Its generality allows to model sensor and action uncertainty, uncertainty in the state of knowledge, and multiple objectives. As such, and under the Markov assumption, it is the holistic model for control, monitoring and repair of complex systems. This is because a solution to a POMDP embodies the optimal chain of actions for all possible beliefs over the world states. This optimal chain of actions is referred to as the optimal policy, solution to the POMDP. If there exists any gain to be made by avoiding or mitigating certain faulty effects, the solution of a POMDP would capture it in its optimal policy.

Another useful advantage of POMDPs is their support of reasoning about whether to select actions that change the state, provide state information, or both. In domains with faults, this proves a key feature since it allows actions to be used to remove ambiguities over the true system state. This is
especially true of actions that carry no utility or cost (mostly used for responding to unlikely situations) and that most planners handle with difficulty since these actions can be inserted anywhere in a plan. In a POMDP framework, any action that changes the state of the world is likely to generate informative observations, and therefore be positioned with accuracy.

Unfortunately despite recent improvements, current solution algorithms still limit the application of the POMDP framework to real world problems. The standard approach to solving a POMDP involves two steps. First the POMDP is transformed into a fully observable Markov decision process with a state space that consists of all probability distributions over the core states of the POMDP. Second, it is solved in this form. For a POMDP with \( n \) core states, the transformed state space is the \( n \)-dimensional simplex, also called belief simplex. This continuous state space is a challenge but can be tackled in practice, either optimally or through approximations. Today, the most promising methods can solve problems with a few thousand states, and yield very approximate solutions to problems with millions of states.

### 1.2 POMDPs for control, monitoring and repair

This paper discusses how to utilize the POMDP framework for control real-world artifacts that are subjected to many unforeseen faulty events. These systems can be characterized as being governed by a number of hard constraints:

- Most actions are potentially risky [Kurien and Nayak, 2000]. Typically an action exhibits a few nominal effects, and a high number of faulty outcomes.
- Many faults, even with low probability of occurrence, can occur anytime. Consequently, they exponentially increase the complexity of the control problem. Such faults are referred to as anytime faults.
- As a consequence the state space is in general orders of magnitude larger than that of currently solvable POMDPs.
- A side effect of anytime faults is that the whole space of faulty states is almost fully reachable from any belief state. This is because the same fault may occur in many different contexts.

This suggests that new approaches could advance the articulation of control, monitoring and repair to a point where it can consider real-world applications with focus on models with faults. This paper is tentative to make the point that the solving of POMDPs can benefit from techniques for diagnosis and efficient state estimation. To this end, it observes that models of the real-world artifacts that contain faults do exhibit a characterized structure:

- Action effects are naturally partitioned into nominal and faulty outcomes.
- Each fault occurrence builds up and becomes more probable over time. Consequently, the probability of faulty states is likely to be higher near the end of an artefact’s life.
- Under an assumption of single fault independence, states that embody multiple faults are expected to be orders of magnitude less likely than others. For this reason the order of a state refers to the number of faults it embodies.
- Under a permanent fault assumption, each state of a given order may lead to states of a limited number of other orders.
- Not all faults can be modeled or are even known. These events are modeled as putting the system into a special unknown state.

This structure may be used to design tailored computational methods capable of handling this class of problems. The applicability of the POMDP framework has been recognized for domains with faults such as ours. However, the structure of the models and problems that are typical of these domains do seem to have been at most briefly considered by the operation research and planning communities. It is not certain that the structural properties mentioned above can generalize to control and planning problems of other domains.

### 2 Previous Work

There has been considerable work on POMDPs, too many to discuss here. But research on efficient computational solutions to POMDPs has made great progress over the past decade.

#### 2.1 Background on POMDPs

We give background on a standard POMDP as found in the literature. The relationship between an agent and its environment is modeled as a discrete-time POMDP with a finite set of states \( S \), a finite set of actions \( A \), and a finite set of observations \( O \). Each time period, the environment is in some state \( s \in S \), the agent takes an action \( a \in A \), and receives a reward for it, with expected value \( r(s,a) \). Taking action \( a \) makes the environment transition to state \( s' \) with probability \( P(s' | s,a) \). In \( s' \), the agent observes \( o \in O \) with probability \( P(o | s',a) \). Let note \( b \) the vector of state probabilities. \( b(s) \) denotes the probability that the environment is in state \( s \). Given \( b(s) \), then after taking \( a \) and observing \( o \),

\[
    b'(s') = \frac{1}{P(o | b, a)} P(o | s', a) \sum_{s \in S} P(s' | s, a) b(s) \quad (1)
\]

where \( P(o | s, a) = \sum_{s' \in S} P(o | s', a) \sum_{s \in S} P(s' | s, a) b(s) \) is a normalizing constant. This belief update is noted \( b' = \tau(b, a, o) \).

The agent’s policy \( \pi \) specifies an action \( \pi(b) \) for any belief \( b \). It is assumed the objective is to maximize the expected total discounted reward over an infinite horizon. Thus the expected reward for \( \pi \) starting from belief \( b \) is defined as

\[
    J^\pi(b) = E \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid b, \pi \right] \quad (2)
\]

where \( \gamma < 1 \) is the discount factor. The optimal policy \( \pi^* \) is obtained by optimizing the long-term reward.

\[
    \pi^* = \arg\max_{\pi} J^\pi(b_0) \quad (3)
\]

where \( b_0 \) is the initial belief.
Exact algorithms for solving this optimization problem are intractable for all but trivial problems. These algorithms solve the Bellman optimality equation that yields the optimal value function,

$$V^*(b) = \max_{a \in A} \left[ r(b, a) + \gamma \sum_{o \in \Omega} P(o \mid b, a)V^*(\tau(b, a, o)) \right]$$

(4)

For finite-horizon POMDPs, the optimal value function is piecewise-linear and convex [E.J. Sondik, 1971]. It can be represented as a finite set of vectors. In the infinite-horizon formulation, a finite vector set can approximate $V^*$ arbitrarily closely, whose shape remains convex. Value iteration applies dynamic programming update to gradually improve on the value until convergence to an $\epsilon$-optimal value function, and preserves its piecewise linearity and convexity. A single dynamic programming pass is often referred to as a backup operation. By improving the value, it implicitly improves the policy as well. Another dynamic programming called policy iteration explicitly represents and improves the policy instead [E. Hansen, 1998].

2.2 Computational solutions

Most of the useful structures and methods have been exploited. As a matter of fact, most solvers now yield very good approximations of the true optimal controls. The approximations are of at least three types: piecewise linear value functions; fixed and variable-resolution grids; belief state space compression. These approximation techniques make use of a set of properties of the POMDP framework. They can be summarized as: reachability analysis; heuristic search; factored representations.

Many algorithms employ a piecewise linear convex representation of the value function and use gradient backups to solve the Bellman optimality equation [E.J. Sondik, 1971; A.R. Cassandra et al., 1997; Kaelbling et al., 1998; E. Hansen, 1998]. These techniques perform gradient backups over the full belief state space. Doing this, the main problem remains the elevated number of vector components, that prevent efficient pruning and linear programming but for a few hundreds of states, at best. Models that include faults have state spaces that are order of magnitude larger, and out of the reach of these representations and techniques.

Another host of algorithms prefer to approximate value functions by focusing their backups onto the relevant belief states instead. The technique is named Point-based dynamic programming. [J. Pineau et al., 2003] present Point-Based Value Iteration (PBVI), an algorithm that scaled the solving of POMDPs up to problems with a thousand states. Point-based techniques in general must select a set of relevant belief points to be backed up. Generation of these points is done through sampling from the action and observation models to generate reachable beliefs. Sampling from a uniform distribution over the full belief simplex has been reported to perform poorly [J. Pineau et al., 2003]. In POMDP models with faults, techniques that make use of reachability suffer from two symptoms. First, as already mentioned, anytime faults make the full state space reachable after a few time steps. Second, point-based techniques would fail backing up the value at most of the low probability states. Recall that most of the state-space is composed of a very high number of low probability states. Therefore missing low probability events inevitably leads to losing a large fraction of the probability mass.

Similar point-based updates can be found in grid-based methods where points form a fixed [W. Lovejoy, 1991] or variable grid over the belief state space [R. Brafman, 1997; R. Zhou and E.A. Hansen, 2001]. While these techniques avoid the pitfalls of stochastic simulation, they do not scale well due to the so-called curse of dimensionality: discretization of the belief simplex scales exponentially with the number of states.

Therefore, another batch of approximated computational solutions try to abstract or compress the POMDP state-space or belief space, respectively. Abstraction uses a reduced (most often factored) representation of the POMDP states in local operations and backups. The background idea is the following: whenever some variables do not affect either some decision of a policy or equivalently, the plan value, they can be marginalized away in the local computations of this policy or value function. Of course the difficulty is to detect these situations at a sufficiently low cost. A popular technique is the exploitation of a factored representation that avoids enumeration of the state-space. [C. Boutilier and D. Poole, 1996] show how to exploit such a representation for POMDPs. An improved version for both value and policy iteration was detailed in [E. A. Hansen and Z. Feng, 2000]. [T. Smith et al., 2007] progresses one step further and builds an abstraction of the factored representation. But the method relies on a structural decomposition of upstream and downstream variables where the former are known by the agent and transition deterministically and the later cannot influence the former. In systems with faults this is almost never the case: stochasticity is at its highest in the upstream variables whereas downstream dynamics are assumed to follow some fault models. Additionally, upstream variables are almost never directly observed.

Compression tries to compact and reduce the belief state space directly. A pioneer work is the dimensionality-reduction technique of [P. Poupart and C. Boutilier, 2002] dubbed Value-directed compression (VDC). It computes a low-dimensional representation of a POMDP directly from its model by finding the Krylov subspace for the reward function under belief propagation. To summarize, it finds the smallest subspace that compresses the value function such that only beliefs that have different values are distinguished. Two drawbacks of this technique is that the Krylov subspace is constrained to be linear and that its computation is as difficult as the exact solving of the underlying POMDP. [P. Poupart and C. Boutilier, 2004] mixes a policy reduction mechanism with VDC and reports on tackling a very large POMDP model of network maintenance (see next section).

[N. Roy et al., 2005] observes that the beliefs are unlikely to lie on a low-dimensional hyperplane. Therefore they propose a non-linear compression scheme, E-PCA. E-PCA is non-linear principal component analysis that extracts a low dimensionality belief space from a heuristic belief distribution. This distribution must be generated for each given prob-
lem. Doing so requires a heuristic controller. In the case of a navigating robot, this means moving the robot around with a heuristic controller and recording the observations. While this technique scales up to problems with a few thousand states, it is not applicable to models with faults. The reason being that a heuristic controller is unlikely to make most faults appear. In other words, it is likely the belief space would compress to its nominal counterpart, leaving most if not all faulty dimensions away.

A class of algorithms combine heuristic (forward) search with dynamic programming backups. [M. Hauskrecht, 1997] describes an algorithm for incrementally calculating the upper and lower bounds a POMDP value function. This bound is initialized with the value for the underlying MDP. It provides a popular way of initializing an upper bound [M.L. Littman et al., 1995], focusing forward exploration [T. Smith and R. Simmons, 2004], heuristic action selection [G. Shani et al., 2007] or value based clustering [Y. Virin et al., 2007]. Heuristic Search Value Iteration (HSV1) [T. Smith and R. Simmons, 2004] has shown some of the best performances yet over large scale problems. HSV1 maintains both an upper and a lower bounds over the value function. It greedily selects belief points through forward exploration and execute backups at the selected points in reversal order on the traversed path. HSV1 avoids the pitfall of sampling the next belief state to be explored and relies on an informed heuristic instead. Its drawback is that it does not compress the belief space so that backups and the computation of its upper bound to the value function are very time consuming and prevent its scaling beyond systems with twenty thousand states or so.

2.3 POMDPs for models with faults

Of all these techniques, we have not found even a handful of them that are reported to have been applied to POMDP models that contain faults. Table 1 report on these works. Among them [B. D’Ambrosio, 1996] is maybe the oldest and the most insightful. It proposes a qualitative compression technique and applies it to a gate-circuit with faults. The author notes how the formulation of diagnosis describes a static, detached process from recovery or maintenance in general. In contrast, he formulates diagnosis as dynamic, practical activity by an agent engaged in an uncertain world, referred to as an on-line maintenance task, a follow-up of his previous work [B. d’Ambrosio, 1992]. Interestingly, he mentions how it is not obvious what elements of a diagnosis are relevant to decision. The gate-circuit model includes an unknown behavioral mode whose behavior is stochastic. [Joshi et al., 2005] formulates a special POMDP whose non directly observable states are fault hypotheses. The optimal policy minimizes the cost of the potential system faults. Of course this work suffers from the size of the problem and falls back on an approximated solution. The sole originality of this solution is the use of a variable horizon: dynamic programming recursion stops whenever the recovered faulty value function falls back within fixed bounds of its sane counterpart. Finally, IM. Hauskrecht, 2001 relies on Monte-Carlo simulation and value function approximations for producing plans with application to the management of patients and medical treatment planning.

3 Examples of control, monitoring and repair of complex systems

This section presents examples of control, monitoring and repair found in the literature and solved by POMDP. These examples have not been designed with monitoring and repair in mind. Thus they lack some key structural properties that are of interest for tackling real world control and repair problems.

3.1 Gate digital circuits

The gate digital circuit example was introduced [B. d’Ambrosio, 1992]. It comes in two flavors: a four gates circuit or half adder, and a seven gates circuit. Within each circuit, a gate has four possible states: ok, stuck-at0, stuck-at1 and unknown. In the latter, the gate output is a stochastic function, independent of the input and uniformly distributed over {0, 1}. Failure probabilities are uniformly distributed over the four states. The agent observes inputs and outputs. Overall, the model is similar to the classical Polybox example. However, the agent can act in several ways: by replacing any component; probing the output of components (in which case the respective value was added to the observation set for the next cycle), or to perform no action. The reward is −1 for each cycle in which at least one component is faulted, −6 for a replacement, and −1 for a probe action. Ignoring the stochastic behavior of the unknown state, the four gates problem exhibits 256 states. As multiple faults are possible, all 256 states are reachable. A policy for this problem anticipates the fault occurrence if the cost of replacing a gate is compensated by the gain of not having several cycles in which at least one component was faulted.

The gate circuit is an interesting model since it is realistic and explicitly models the unknown state of behavior. However, its replacement action is deterministic. Also, there is no wear to the board, so the fault rate remains constant (e.g. there is no modeled wear to the board).

3.2 Machine Maintenance of Cassandra

The machine maintenance problem is an early but difficult application model for POMDPs [Cassandra, 1998]. The model

Table 1: Approximation techniques and fault models.

<table>
<thead>
<tr>
<th>Faults</th>
<th>Piecewise linear convex grid</th>
<th>heuristic search compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults</td>
<td>[B. D’Ambrosio, 1996]</td>
<td>[E. A. Hansen and Z. Feng, 2000], [T. Smith and R. Simmons, 2004], [G. Shani et al., 2007]</td>
</tr>
<tr>
<td></td>
<td>[Joshia et al., 2005]</td>
<td>[P. Poupart and C. Boutilier, 2002], [B. D’Ambrosio, 1996]</td>
</tr>
</tbody>
</table>
is very generic: it describes a machine made of $c$ internal components. The machine is used to produce a product. The quality of the product is a function of the state of the internal components. Every component is modeled by four possible states: $good$, $fair$, $bad$ (it can be repaired) and $broken$ (it must be replaced). As each components has four possible states, the problem has $4^c$ states. Components can only be observed if the machine is disassembled: this requires an expenditure of time and personnel while rendering the machine unproductive for the duration of the inspection.

The agent seeks the maintenance of the machine and can utilize a finite set of actions. The $manufacture$ action proceeds with production while each component deteriorates with probability 0.03 after each day. The only possible transitions are from $good$ to $fair$, from $fair$ to $bad$ and from $bad$ to $broken$. This is represented Figure 1. The $inspect$ action does not change the underlying state but yields information about the hidden state, see below. The $repair$ action improves every component’s condition with probability 0.8. Possible transitions are from $fair$ to $good$ and $bad$ to $fair$. A $broken$ component cannot be repaired. The $replace$ action deterministically puts a whole new set of components in good condition. Taking the $manufacture$ action allows to observe the quality of the products, either $good$ or $bad$. A $good$ component always performs properly during the day. A $fair$ component has probability 0.95 of performing properly, while a $bad$ component has probability 0.75. A $broken$ component never performs properly. Taking the $inspect$ action, the observation is a composite of individual observation for each component, each of which is observed to be in a either $good$ or $bad$ condition. The probabilities depend on the actual condition of the component. A $good$ component will yield a $good$ observation with probability 0.97, a $fair$ component looks $good$ with probability 0.80, $bad$ with probability 0.05 and $broken$ with probability 0.02. There is no observation for the $repair$ and $replace$ actions. The agent is rewarded every day (1 for each good product). The $inspect$ action costs $-1$. Repair costs $-3$. Replacing the machine costs $-15$.

This example is very complete, generic, and can be easily extended with other components. However it does not model wear since the failure rate remains constant.

3.3 The network maintenance example

This example is presented in [P. Poupart and C. Boutilier, 2004]. A system administrator (SA) maintains a network of machines. Each machine has a 0.1 probability of failing at any stage; but this increases to 0.333 when a neighboring machine is down. The SA only observes the status of a machine (with 0.95 accuracy) if he reboots or pings it. At each stage, he can either reboot or ping a machine, or do nothing. The SA receives a reward of 1 per working machine and 2 per working server. Costs are 2.5 (rebooting), 0.1 (pinging), and 0 (doing nothing). An $n$-machine network induces to a POMDP with $2^n$ states, $2n + 1$ actions and $2n$ observations. This example is scalable and can exhibit over 33 millions states (for 26 machines). Of interest here is the failure rate, that is no more constant. In fact it depends on the topology of the network. The model comes in two flavors: a ring topology in which each computer communicates with at most two neighbors; a star-like topology: a tree of three branches joined at the root. While this is somehow realistic, the failure rates remain constant over time.

3.4 A novel example: the taxi maintenance

We introduce a novel example that embodies the most realistic features of a control, monitoring and repair problem. The taxi maintenance problem builds on Cassandra’s machine. A driver gets money each day by driving its taxi around. His car is made of several critical parts, and is modeled has a machine with $c$ components. In addition to the component states, the taxi maintenance models the age of the car. The taxi maintenance problem has several important features:

- the failure rate of every component is a function of the age of the car;
- the car ages faster when driven with one or more bad or broken components;
- the decision problem ends whenever the car must be replaced.

The car wears normally everyday so its age increases linearly with time. But driving the taxi with some faulty internal components wears the car faster. Very importantly, the failure rate is an increasing function of the car’s age. Here, the fault probability distribution of every component is modeled by a Weibull distribution with a shape parameter $> 1$ [Weibull, 1951]. The model considers the problem over a finite horizon that is a number of days $d$. The true age of the car is bounded by $1 \leq d$ (after which it must be replaced). The taxi maintenance problem can be seen as a machine maintenance problem for every possible age value of the car. Thus the number of states is $14^c$.

The taxi maintenance problem poses a certain number of problems to existing POMDP techniques. First, approximation methods that select the most likely belief states would lose track of the true car’s age. The consequence is a disastrous cascading effect: as age determines the true failure rates, future failure rates are wrongly estimated, augmenting the drift from the true car’s age. Second, due to potential large values of $l$, the state-space is very large: considering the problem over five years yields 1825 days, and 467200 states. Third, since the car’s age remains a function of time, the belief over the true car’s age is pushed forward over time. In consequence, compression techniques able to mitigate the high number of states would need to leverage on reachability and be dynamic over time to remain efficient.
4 Approaches

The idea defended here is that solution approaches could benefit from a diagnostic and repair stance on the monitoring and control problem. Thus some of the intuitions in the DX community that led to good solution techniques could act as the root basis for targeted approaches to the solving of POMDPs.

4.1 Belief point selection

Typically, the control of a POMDP is decomposed into two parts: a state estimator that performs the belief update of equation (1); a planner, that finds the optimal policy (4). State estimation of models with faults is a topic that has been successfully tackled by the Control, FDI and DX communities. Of special interest here, the blowup in state estimates of complex models modeled as discrete and hybrid systems has received a large attention. Solution techniques employ reasoning and statistical methods that leverage the number of estimates. Two examples from the literature are worth considering.

[Hofbaur and Williams, 2004] mitigates the number of state estimates at each time step. It utilizes an A*-like search procedure with an information-based heuristic to select the set of most likely estimates. A similar strategy can be employed in POMDP solution techniques. The difference is that all observations must be considered. Thus a forward search procedure would retain a fixed size set of most likely combination of actions and observations. In a way this is similar to HSVI’s expansion phase that produces trajectories within the belief space. But the generated traversals carry single belief points. Instead, the proposed extension would select and project sets of most likely points.

[Verma et al., 2003] introduces a variable resolution particle filter. The resolution of state space is dynamically varied by region, depending on the belief that the true state lies within a region. Where the belief is strong, the resolution is fine, where the belief is low, the resolution is coarse, abstracting multiple similar states together. This abstraction is predefined while the resolution results of a bias-variance trade-off. Taking this mechanism to POMDP is akin to a variable resolution dynamic compression of the state-space. The underlying intuition however slightly differs. Where in the belief simplex the belief is strong, i.e. at and around its corners, the resolution is coarse. This reflects that the statistic contains enough information for near optimal control. The resolution is fine in and around the pit of the belief convex hull. This is because in this belief region the agent is very uncertain about where the true underlying state of the world: selection of an appropriate action demands a reduced bias.

4.2 Simple rank-based compression

In a way, uncertainty in action outcomes can be viewed as a halo of unwanted, sometime unforeseen, consequences that surround a desired outcome. This hoped-for outcome is here dubbed nominal. It can be modeled as having a higher probability of occurrence than other outcomes, a reflection of the original (or nominal) intention hidden behind the action that produces it. Often it is simply modeled as the mean expected outcome. Note that a consequence is that the value of states with high rewards often “trickle down” to neighboring states [Y. Virin et al., 2007]. Under an assumption of single fault independence, it is clear that states that embody deviations from the nominal outcomes become order of magnitude less likely than the nominal state over time. Of course in certain cases deviation effects can cancel out. This is akin to the case of non exoneration of multiple faults in diagnosis.

So an approach is to utilize the notion of rank [M. Goldszmith and J. Pearl, 1996] for representing an order of magnitude probability scale for faults. The nominal states are said to be of rank 0. Failure states are one or more orders of magnitude less likely and thus of ranks 1 to n. This qualitative representation also acknowledges the fact that real probabilities are often either inaccessible or unknown. Thus the notion of rank can be considered as an expression of the degree of doubt.

Assume a POMDP models n individual component faults. It is possible to abstract this POMDP state space according to ranks and to build a compressed POMDP over the ranks. This rank-based POMDP has at least $2^n + 1$ states: all possible faults combinations plus at least one nominal state. Let note the compressed state space $S = \{S_1, S_2, \cdots, S_n\}$ where $S_k$ is the abstract state of rank k. Each $S_k$ contains all original states that model exactly k faults. $S_k$ contains $\binom{n}{k}$ fault combinations.

Within this compressed state space, each action has a nominal outcome of rank 0, and some deviation outcomes of ranks ranging from 1 to n. Considering a single fault per time step, action $a \in A$ outcomes can be compressed into at most $2n + 1$ compressed outcomes: n faulty, n repair, and one nominal outcomes. Thus actions for the rank-based POMDP are reduced to simple jumps among compressed states of ranks: k to k+1 in case of a fault occurrence; k to k−1 in case of fault repair; k to itself if nothing happens (e.g. nominal outcome realized from a faulty state). The compression of actions easily extends to $d \leq n$ faults per time step. Figure 2 pictures a compressed automaton for such a POMDP whose actions can all be compressed into faulty outcomes of deepest rank d at each time step, and a one step repair outcome, of respective probabilities $p_f$ and $p_r$. Note how the Machine Maintenance example is a variant of this model with a single fault per time step, and a repair action outcome is a jump over up to four ranks. Both the observation and reward function are compressed over S. Simply, $R(S_k, a) = \sum_{s \in S_k} R(s, a)$.
4.3 Rank-based dynamic compression

The compression scheme above regroups faults based on their likelihood of occurrence. In consequence it cannot distinguish among faults. In general this is bad for decision since some faults can have more deadly effects than others. So the agent lacks proper discernment and chooses actions greedily, oblivious of their true effect. Applied to the network maintenance example, this means that the agent would not be able to distinguish among the failed machines, but only among the number of failed machines. Applied to a star topology like the one on this same network problem, the strategy may be a compression that yields a useful approximated policy: it often makes sense to reboot the central machine so the ability to distinguish among all machines is not essential. What matters is for which belief level does it make sense to reboot this server machine. Applied to a ring topology however, the result is potentially disastrous: the best policy available to the agent would be something akin to randomly choose and reboot a machine. This is because the granularity of available observations (through pinging a machine) is abstracted away by the rank-based compression.

In practice this effect is slightly mitigated because the rank-based compression is dynamic. Dynamic compression is policy dependent. This means that the states in each S_l when following a policy are those states of rank k that are reachable by this policy. This is useful since some faults might simply never occur when certain chains of actions are taken. For example, regularly repairing an equipment of its taxi allows the driver to almost never experience a fault on this equipment. However, the approach is not scalable since the compression into ranks is ad-hoc and domain dependent. For example, it does not extend to robot navigation problems often found in the POMDP literature.

A more general approach starts from [M. Goldszmith and J. Pearl, 1996] and writes beliefs as polynomials of the form
\[ b(s) = \sum_{k=1}^{n} b_k(s) \epsilon^k \]
where the number is infinitesimal. The compression scheme builds a compressed belief \[ \tilde{b}_k = \sum_{s \in S_k} b_k(s) \epsilon^k. \] This compression is lossy but \( \epsilon \) and \( n \) control its accuracy w.r.t. the original POMDP. Error carried by a compressed belief point is bounded by \( \epsilon^n. \) The compressed POMDP has now \( n \) states. The \( b_k \) define a basis for the vector space of the value function. Importantly, compression is dynamic: \( b_k \) and \( \tilde{b}_k \) at different time steps define different basis. This means an adaptive compression is applied at each time step that conserves the order of magnitude relatively among the states. The bad news is that the value function is now expressed in a different basis at each time step. This means the vectors that compose it must be transformed from one basis to the other during backup operations. The transformation function is a compressed version of the action model of the original POMDP.

Overall, this approach is close to the qualitative belief space described in [B. d’Ambrosio, 1996]. However, [B. D’Ambrosio, 1996] applies a Kappa calculus point-based abstraction to a model-free POMDP. Thus it does not carry a full representation of the value function as a set of vectors, and does not compress the action model, but learns it from a set of value points instead. The main difference of the outline approach w.r.t. VDC is that it is dynamic and leverages on the reachability of the belief states. Thus compression near the end of a policy abstracts away unreachable states. In our application domain, this means more weight is given to the faulty events that are more likely to occur over time.

Conclusion

This paper has discussed both the advantages and the challenges of modeling control, monitoring and repair problems as POMDPs and solving them. A novel example is given, the taxi maintenance problem, that embodies most of the features of real world control problems for artifacts subjected to wear and multiple faults. Research on POMDPs remains a very active field. Many recent advances allow to close the gap between reality and its problems on one side, and representation and computational techniques on the other side. Let us mention a few more very recent works in decision and control in partially observable domains and that should be of interest to the DX and FDI community. In [H. Itoh and K. Nakamura, 2007] is presented a version of the POMDP framework with imprecise (intervals) parameters. [S. Ross et al., 2007] similarly allows uncertain parameters but models and learns them with Dirichlet distributions. Finally, [E. Brunskill et al., 2008] mixes a dynamical model of a system’s dynamics (in the form of differential equations) with the standard POMDP framework, in what we believe is a necessary step for the robust and intelligent control of future machines.

References


Generating Diagnoses from Conflicting Component Sets with Continuous Extents

Emmanuel Benazera and Louise Travé-Massuyès
LAAS-CNRS, Université de Toulouse, 7 av. du Colonel Roche, 31077 Toulouse Cedex 4, France. email: ebenazer,louise@laas.fr

Abstract

Many techniques in model-based diagnosis and other research fields find the hitting sets of a collection of sets. Existing techniques apply to sets of finite elements only. This paper addresses the computation of the hitting sets of a collection of sets whose elements have convex or non-convex, bounded or unbounded continuous attributes, also referred to as continuous extents. We assume the conflict sets are known and we present a novel procedure, the Continuous Hitting Set algorithm (CHS) for transforming conflict sets of elements with continuous extends into minimal hitting sets.

1 Introduction

Many theoretical and practical problems can partly reduce to an instance of the minimal hitting set problem, or its close variant the minimum set covering problem. One widely recognized application is in the field of model-based diagnosis [Reiter, 1987]. In this approach, a system is a tuple \((COMP S, SD, OBS)\); \(COMP S\) is a finite set of system components; \(SD\) is the system description; \(OBS\) is the observation. A diagnosis is a minimal set \(D \subseteq COMP S\) such that under the assumption that all other components are behaving correctly, \(D\) explains the observation given \(SD\). In the diagnosis community, \(D\) is said to be consistent with \(SD\) and \(OBS\). This approach to diagnosis has two steps: (i) a collection of all minimal conflict sets is computed; (ii) the conflict sets are transformed into diagnoses. A conflict set \(s \subseteq COMP S\) is such that the assumption that all components in \(s\) are behaving correctly is not consistent with \(SD\) and \(OBS\). A minimal conflict set is such that it does not contain any other conflict set. Reiter showed that the minimal diagnoses are the minimal hitting sets of the collection of minimal conflicts.

Since the beginning of model-based diagnosis, several algorithms for computing the hitting sets have been introduced. Most rely on the building of a so-called HS-DAG [Greiner et al., 1989] or HS-tree [Wotawa, 2001] but other representations exist [Haenni, 1997; Lin and Jian, 2003]. All these techniques transform conflict sets of discrete elements into diagnoses. But in many applications of model-based diagnosis, the conflicts contain more information. This information includes but is not limited to intervals of possible failure time in systems with functional delays, or continuous parameter ranges found in fault models. For example, in systems with delays, several conflicts may involve the same components with different estimates of the symptom occurrence dates [Travé-Massuyès and Calderon-Espinoza, 2007].

In this paper, we address step (ii) and assume all minimal conflicts are given. Each conflict element is a component with associated bounded or unbounded intervals over a continuous line. We assume there is a single continuous attribute per component, but this assumption has no incidence on the generality of the presented method. The problem with conflicting component sets with continuous attributes is that minimal diagnoses are conditioned upon the continuous values. This is because a minimal diagnosis corresponds to a minimal continuous region. A diagnosis in this context is a set of \(k\) components along with a set of bounded or unbounded regions of \(\mathbb{R}^k\). Existing algorithms are not designed to find and construct these regions. A naive strategy would be to apply these algorithms to a collection of conflicts with selected elements of the continuous lines. However, since the hitting set problem has a worst case performance that is exponential the size of the collection of conflicts, this would hardly prove an efficient approach. Moreover, many points that belong to the same minimal diagnoses would be computed independently.

This paper presents a general computational method for finding the hitting sets of a collection of conflicts with continuous attributes. The algorithm is named CHS for Continuous Hitting Sets. Starting from the classical approach, the proposed solution searches the hitting sets in an aggregate space of diagnoses. Similarly to the classical methods the CHS has both an expansion and a pruning phases. It is shown how the pruning phase dominates the computational effort. Simulation experiments on hundreds of randomly generated conflicts assess the main properties and the scalability of the CHS.

2 Problem Definition

2.1 Conflicts and diagnoses

We consider a tuple \((COMP S, SD, OBS)\). \(COMP S\) is the set of physical components composing the physical system. \(OBS\) is a set of observations. Given \(OBS\), the diagnosis operation derives all sets of faulty components of \(COMP S\) that may explain the facts. In the sense of Reiter, the conflict
sets, or conflicts for short, are the sets of components which 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. Its main result being that minimal hitting sets of a collection of minimal conflicts yield the minimal diagnoses, where a hitting set of a collection of sets is a set intersecting every set of this collection.

Theorem 1 ([Reiter, 1987], Th. 4.4). \( D \in \text{COMP S} \) is a diagnosis for \((\text{COMP S}, SD, OBS)\) iff \( D \) is a minimal hitting set for the collection of conflict sets for \((\text{COMP S}, SD, OBS)\).

The relation above was established for conflicts as sets of components with no attributes. Each component in a diagnosis hence belongs to one or more conflicts. We say a component explains, or equivalently covers these conflicts.

**Hitting set algorithm**

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 et al., 1989]. This algorithm gives a means to compute diagnoses incrementally, under the permanent fault assumption.

The diagnosis algorithm builds a Hitting-Set tree (HS-tree) in which all the nodes but leaves are labelled by a conflict set. For each component \( C \) in the conflict label of node \( \tau \), an edge labelled \( C \) joins \( \tau \) to a successor node. \( H(\tau) \) is defined as the set of edge labels on the path from \( \tau \) to the root node. The HS-tree is built by considering every conflict in arbitrary order. Every new conflict is compared to every leaf of the HS-tree, and some new leaves are built if necessary. The resulting HS-tree is pruned for redundant or subsumed leaves before the next conflict is considered. Pruned leaves are said to be closed. At the end of the diagnosis procedure, the minimal hitting sets, and hence the minimal diagnoses that explain the system’s misbehaviors, are given by the set of edge labels \( H(l) \) associated to the open leaves \( l \) of the HS-tree.

**A water transport system example**

We consider the simple example of a two reservoirs system, pictured on Figure 1. A system consists in continuously supplying water to two consume areas \((o_1 \text{ and } o_2 \text{ are the corresponding flows})\) from two cascaded geographically distant reservoirs \( C_1 \) and \( C_2 \) \((y_1 \text{ and } y_2 \text{ are the water levels in the respective reservoirs})\). The water transport between reservoirs is modeled as an open flow channel. A hand-switch regulates the water pressure by channeling it either through a pipe \( C_3 \), or through a pump \( C_4 \). Assume that \( o_1 \) and \( o_2 \) can be measured. In the full application, the system is modeled by a set of discrete time equations that need not to be presented here. So \( \text{COMP S} = \{C_1, C_2, C_3, C_4\} \) and \( \text{OBS} = \{o_1, o_2\} \).

**Example 1** Assume \( o_1 \) is measured and found discrepant while water is channeled through \( C_3 \). This produces conflict \( s_1 = \{C_1, C_2, C_3\} \). Similarly, assume \( o_2 \) is measured and found discrepant while water is pumped through \( C_4 \). This discrepancy leads to conflict \( s_2 = \{C_1, C_2, C_4\} \). This problem has three minimal diagnoses given by the hitting sets for \( \{C_1, C_2, C_3\} \) and \( \{C_1, C_2, C, C_4\} \): \( D_1 = \{C_1\}, D_2 = \{C_2\}, D_3 = \{C_3, C_4\} \).

**2.2 Continuous generalization of conflicts and diagnoses**

In many real-world applications, the conflicts contain more information than the components. Thus an attribute to a component \( C \) in a conflict can be any additional piece of information, either temporal, such as \( C \) has been faulty for at most four units of time, or functional, such as \( C \) is faulty with a certain parameter in value range \([a, b]\). Taking such information into account necessitates an extension to the existing framework.

**Continuous extent**

To begin with, [Dressler and Freitag, 1994] define the temporal extent \( TE(\alpha) \) of a proposition \( \alpha \) as the set \( \{t \mid \alpha \text{ holds at } t\} \). Following the literature, let us denote the fact that \( C \in \text{COMP S} \) is faulty by the predicate \( AB(C) \). Then \( TE(AB(C)) \) is the set of time points at which \( C \) is faulty. Unfortunately this approach is limited to points in a continuous space. In application, attributes come in more general forms. Consider a continuous space \( X \) of finite dimension. We adopt a more general approach to modeling continuous information in the model-based diagnosis framework. In this novel formulation, a proposition attribute can be any point, interval (bounded or unbounded) or set of intervals (convex or non convex), within \( X \). We name this extension the continuous extent \( CE(\alpha) \) of a proposition \( \alpha \).

**Definition 1** (Continuous extent). The continuous extent of a proposition \( \alpha \), \( CE(\alpha) \), denotes the continuous region \( \{\Omega \mid \alpha \text{ holds in } \Omega\} \).

We use the following notation: given an interval \( I \subset X \), \( C_I \) is a short way to represent \( CE(AB(C)) \subset I \).

**Example 2** Consider again the example on Figure 1, and consider the additional temporal information. \( \tau_1 \) between the two reservoirs and \( \tau_2 \) between the pump and reservoir 1 are the transport time delays, with \( \tau_1 \geq 0, \tau_2 \geq 0 \). Water traverses a reservoir in 1 unit of time. In this example, the information on the failure time of a component \( C \), relatively
to the current time, is a continuous extent of the proposition \( AB(C) \).

**Conflicts and diagnoses with continuous extents**

In context, each component \( Ci \) of \( COMPS \) operates over a continuous line \( x_i \), where \( i \) is the component index. This line is a continuous extent line that supports the component failure time, or any fault related parameter or state variable values. In case where it is bounded, it is referred as the continuous extent domain of \( Ci \). Thus \( X = \bigotimes_{i=1}^{M} x_i \), where \( M \) is the number of components in \( COMPS \). Here we assume there is a single continuous extent line \( x_i \) per component \( Ci \). This assumption allows to simplify notations, but it has no incidence on the generality of the formalism and the computational method proposed in this paper.

We assume component \( Ci \) in a conflict \( s \) has a known uni-dimensional failure interval \( I_i^s \subseteq x_i \). Following definition 1, \( C_i \) denotes a failure of \( Ci \) within interval \( I_i^s \). To simplify notations \( C_{ij} \), where \( j \) is a real or an integer denotes \( C_{i[j, \infty)} \). A conflict of cardinality \( k \) defines a continuous region \( \bigotimes_{i=1}^{k} I_i \) that is a hypercube of \( \mathbb{R}^k \). Given a collection \( S \) of conflicts, a diagnosis of cardinality \( k \) is a tuple \((D, X)\) where \( D \) and \( X \) are the discrete elements and associated continuous extents, respectively. Thus a minimal diagnosis \( D = \{C_1, \ldots, C_k\} \) with continuous extent defines a minimal continuous region \( X = \bigotimes_{i=1}^{k} I_i \) that is a hypercube of \( \mathbb{R}^k \). This region is equivalently referred to as the minimal continuous extent of diagnosis \( D \), or a minimal continuous diagnosis.

**Example 2** (continued). Assume \( \alpha_1 \) is measured and found discrepant while water is channeled through \( C_3 \). By tracing back the system’s delays, this produces conflict \( s_1 = \{C_{11}, C_{27,31}, C_{37,31}\} \). This is because \( \alpha_1 \) discrepant implies that \( C_3 \) was faulty at least a unit of time ago, and/or \( C_2 \) and/or \( C_3 \) at least \( 2+1 \) units of time ago. Similarly, assume \( \alpha_2 \) is measured and found discrepant while water is pumped through \( C_4 \). This discrepancy leads to conflict \( s_2 = \{C_{14,12}, C_{24,14}, C_{47,37,32}\} \). Then, \( (D_1, X_1) = (\{C_1, x_1 \in [\tau_1, 1, +\infty]\} \) and \( (D_2, X_2) = (\{C_2, x_2 \in [\tau_2, 1, +\infty]\} \) are two diagnoses. \( (D_1, X_1) \) assumes \( C_3 \) has been faulty long enough, i.e. for at least \( \tau_1 \) units of time, so that it can explain both conflicts \( s_1 \) and \( s_2 \). \( (D_3, X_3) = (\{C_1, C_2\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_2, 1, +\infty]) \) assumes that (i) \( C_1 \) has not been faulty for long enough so that it can explain \( s_2 \); (ii) \( C_2 \) has not been faulty for long enough so that it can explain \( s_1 \). Overall, there are three more diagnoses that in Example 1:

\[
\begin{align*}
(D_4, X_4) &= (\{C_1, C_4\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_1, 1, +\infty]) \\
(D_5, X_5) &= (\{C_3, C_2\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_1, 1, +\infty]) \\
(D_6, X_6) &= (\{C_3, C_4\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_1, 1, +\infty])
\end{align*}
\]

**Proposition 1.** Given a collection \( S \) of conflicts, a diagnosis \((D, X)\) is such that

\[
X \subseteq \bigotimes_{i=1}^{k} \bigcup_{I_i^s \in S_i} I_i^s
\]

where \( S_i^s \subseteq S \) is the subset of conflicts that are explained by \( Ci \).

**Proof.** Consider a collection \( S \) of conflicts. Then for every component \( Ci \in D \), there are three non-exclusive cases:

1. It exists at least \( s \) st. \( I_i^s \subseteq I_i^s \), for all \( s' \in S \). In this case, \( Ci \) explains all conflicts in \( S \) with \( I_i = \bigcap_{s \in S_i^s} I_i^s \) and \( S_i^s = S \).

2. \( \exists s \subseteq S, s' \subseteq S \) st. \( I_i^s \neq I_i^s \) and \( I_i^s \cap I_i^s = I \) with \( I \neq \emptyset \). In this case

\[
\begin{align*}
C_i \text{ explains } \{s \} &\text{ for } I_i = I_i^s \text{ and } S_i^s = \{s\}, \\
C_i \text{ explains } \{s' \} &\text{ for } I_i = I_i^s - I \text{ and } S_i^s = \{s'\}, \\
C_i \text{ explains } \{s, s' \} &\text{ for } I_i = I_i^s - I \text{ and } S_i^s = \{s, s'\}.
\end{align*}
\]

3. \( I_i^s \) are all disjoints for all \( s \in S \). In this case, \( Ci \) explains conflict \( s \) iff \( I_i = I_i^s \) and \( S_i^s = \{s\} \).

It follows that minimal diagnoses are either such that \( I_i = \bigcap_{s \in S_i^s} I_i^s \), in case 1 and 3, and \( I_i \subseteq \bigcap_{s \in S_i^s} I_i^s \) in cases 2.

Since \( X = \bigotimes_{i=1}^{k} I_i \), it comes (1).

The relation (1) is a consequence of having conflicts with overlapping continuous extents. This proposition conditions diagnoses upon regions that are smaller than the continuous extent in each conflict.

**Example 2** (continued). Consider diagnosis \( (D_3, X_3) = (\{C_1, C_2\}, x_1 \in [\tau_1, 1, +\infty]) \) for the conflicts \( s_1 = \{C_{11}, C_{27,31}, C_{37,31}\} \) and \( s_2 = \{C_{14,12}, C_{24,14}, C_{47,37,32}\} \). Proposition 1 applies: \( C_1 \) explains \( s_1 \), and \( C_2 \) explains \( s_2 \), with \( x_1 \in [\tau_1, 1, +\infty] \) and \( x_2 \in [\tau_2, 1, +\infty] \).

Reiter’s main result applies to the extended representation.

**Theorem 2.** \((D, X)\) is a diagnosis for \((COMPS, SD, OBS)\) if \( D \) and \( X \) are minimal hitting sets for the collection of conflict sets with continuous extent for \((COMPS, SD, OBS)\).

**Example 2** (continued). There are six diagnoses that are the hitting sets for the conflicts with continuous extents \( s_1 = \{C_{11}, C_{27,31}, C_{37,31}\} \) and \( s_2 = \{C_{14,12}, C_{24,14}, C_{47,37,32}\} \):

\[
\begin{align*}
(D_1, X_1) &= (\{C_1, x_1 \in [\tau_1, 1, +\infty]\}) \\
(D_2, X_2) &= (\{C_2, x_2 \in [\tau_2, 1, +\infty]\}) \\
(D_3, X_3) &= (\{C_1, C_2\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_2, 1, +\infty]) \\
(D_4, X_4) &= (\{C_1, C_4\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_1, 1, +\infty]) \\
(D_5, X_5) &= (\{C_3, C_2\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_2, 1, +\infty]) \\
(D_6, X_6) &= (\{C_3, C_4\}, x_1 \in [\tau_1, 1, +\infty], x_2 \in [\tau_2, 1, +\infty])
\end{align*}
\]

Here diagnosis \( \{C_1\} \) explains both conflicts iff \( x_1 \in [\tau_1, 1, +\infty] \). Otherwise, \( \{C_1\} \) cannot explain \( s_2 \). Therefore an additional component is needed for a diagnosis to explain the conflicts when \( x_1 \in [0, \tau_1] \). Also, the attentive reader notices that \( \{C_1\}, \{C_2\}, \{C_3, C_4\} \), the diagnoses previously obtained on Example 1 with no continuous extent are present in the solution above. However, three new diagnoses have

been found: \{C_1, C_2\}, \{C_1, C_4\}, \{C_3, C_2\}. Obviously, these diagnoses are the consequence of the presence of additional information in the form of continuous extents. To each minimal diagnosis corresponds a minimal region of \(\mathcal{X}\). By characterizing diagnoses with continuous extents, Theorem 2 provides the basis for the computation of diagnoses. Therefore, finding the minimal diagnoses and associated minimal continuous regions is the aim of the continuous extension to the HS algorithm that is presented in the next section.

## 3 Generating Diagnoses from Conflicts with Continuous Elements

### 3.1 Solution Approach

The original hitting set algorithm considers conflict sets with discrete elements only. It looks for singletons in each conflict set. The algorithm cannot condition the diagnoses upon the different continuous failure points of a component. As briefly shown in the previous section, doing this significantly enlarges the number of diagnoses. It follows that the difficulty we address in this paper is the potentially huge size of the space of diagnoses over continuous regions. The reason for this size is the existence of continuous variables. The hitting set algorithm is exponential in the number of conflict elements so the number of potential diagnoses is staggering.

Underlying the diagnoses are the conflicts, each being explained by the failure of a component in certain regions of its continuous line. It follows that the dimension of the continuous space is the total number of different components in the set of conflicts. In general we assume the continuous space dimension to be equal to the number of components in the system. The challenge is thus to apply the hitting set algorithm to this continuous state-space. Our solution to address this issue is to search for minimal diagnoses in an aggregate space of diagnoses. This space is represented by a directed acyclic graph (DAG) in which there is a node for each potential diagnosis discrete element. In other words, each node of our DAG represents a continuous region in which the discrete diagnosis element is the same. Given such a partition of the continuous space, the CHS uses a DAG structure to build the minimal continuous diagnoses that are the hitting sets of the collection of conflicts with continuous extents.

To take advantage of the aggregated representation of the space, the standard HS algorithm must be modified in important ways. In particular, there is no longer a unique correspondence between a node and a diagnosis \((D, X)\). Each node of the DAG now supports the continuous diagnoses in \(X\) that have an identical discrete element \(D\). A consequence is that different conflicts can be explained in different regions of a node’s continuous region. To address this problem and find the hitting sets, the nodes of the DAG receive a set of functions of the continuous space \(\mathcal{X}\) that allow to map different conflicts to different continuous regions. This reflects in the more complex data structures required by the CHS.

In summary, a simple way of understanding the CHS algorithm is as a variant of the HS algorithm where for every conflict, candidate diagnoses with identical discrete elements are expanded in unison. The main difference with the HS is threefold:

- In the standard HS-tree, a single diagnosis is associated with each node. In the CHS, multiple diagnoses are associated with a single node.
- The CHS produces a DAG instead of a tree.
- Nodes are often simultaneously a leaf and a node in the interior of the DAG. This is a consequence of having different conflicts explained in different continuous regions of the same node. In particular, this happens when a part of the aggregated diagnoses do explain all conflicts, while another part does not.

### 3.2 Data Structures

The main data structure represents a node \(n\). Given a set of conflicts \(S\), it contains:

- A diagnosis \(H(n)\) that is a set of \(k_n\) edge labels, i.e. components.
- A region \(X_n\) of continuous diagnosis elements. It represents the continuous lines of the components in \(H(n)\), \(X_n \subseteq \mathbb{R}^{k_n}\).
- \(\text{Open}_n() \rightarrow \{0, 1\}\): the Open function. For each \(x \in X_n\), \(\text{Open}_n(x)\) indicates whether \((n, x)\) explains all conflicts in \(S\). The open region of \(n\) is noted \(\Omega_n = \{x \in X_n | \text{Open}_n(x) = 1\}\). A diagnosis is either open or closed. Note that we don’t refer to open or closed nodes; instead we refer to diagnoses associated with nodes as being open or closed.
- \(\delta_n(\ldots)\) the explanation function. For \(x \in X_n\) and \(s \in S, \delta_n(s, x)\) indicates whether \(s\) is explained by \((n, x)\).

Formally,

\[
\delta_n(s, x) = \begin{cases} 
1 & \text{if } \exists C_i \in I^* \text{ st. } C_i \in s \cap H(n) \text{ with } x \in I_i^*, \\
-1 & \text{otherwise}.
\end{cases}
\]  

(2)

### 3.3 The CHS Algorithm

Algorithm 1 presents the main procedure.

**Expansion (lines 9 to 12):**

For a node \(n\) and a conflict \(s\):

\[
A_n(s, x) = \begin{cases} 
\{C \in s | C \notin H(n)\} & \text{if } \delta_n(s, x) = -1 \\
\emptyset & \text{otherwise}
\end{cases}
\]

(3)

is the set of discrete conflict elements that can expand \((n, x)\). At each iteration, CHS expands a diagnosis \((n, x)\) if it doesn’t explain the conflict \(s\). An important distinction between HS and CHS is that in the latter, nodes are often partially expanded. This means not all conflicts are explained by some diagnoses \((n, x)\) of node \(n\). The catch is that only those \((n, x)\) that do not explain all conflicts are expanded, and closed after expansion.

**Computing the explanation functions (lines 10 & 4):**

Each newly expanded \((n, x)\) must be updated. This consists in recomputing its explanation function (Eqn (2)).
Opening & closing of continuous regions (line 10):
The algorithm proceeds by leaving open the regions of the continuous space that explain all conflicts, and by closing the others. Similarly to the original HS, the CHS has an expansion phase and a pruning phase. In the expansion phase, \((n, x)\) is closed if it has been expanded, or if \(\exists s\) st. \(\delta_n(s, x) = -1\) and \(A_n(s, x) = \emptyset\). In the pruning phase, \((n, x)\) is closed if it is subsumed by some other node \((n', x')\) such that \(H(n') \subseteq H(n)\) and \(\Omega_{n'} \subseteq \Omega_n\). For every new conflict \(s\) and every element \(C\) of the conflict, the algorithm builds two lists, newleaves\([C]\) and oldleaves\([C]\), which are then compared. Closed regions of a given node cannot be reopened. This is easily seen since closed regions contain points that do not explain all conflicts. Therefore these regions are expanded into new nodes. The \((n, x)\) that remain opened after all conflicts in \(S\) have been processed are the minimal diagnoses.

Example:
Consider the two conflicts of example 2.1. Figure 2(a) pictures the CHS structure after the expansion of \(s_1\). Expansion of \(s_2\) leads to the closing of the subregion 1 \(\leq x_1 < \tau_1 + 1\) of node 1, and closes node 3, see Figure 2(b). Node 2 is unchanged since after step 4, \(\delta_2(s, x_2) = 1\) for all open \(x_2 \geq \tau_2 + 1\), leaving \(A_2()\) empty. The pruning phase closes nodes and regions. A node \(n\) is closed whenever for all \(x\), \(\Omega_n(x) = \emptyset\) for all \(x \in X_n\). Node 6 is closed as it is subsumed by node 1. Similarly, node 2 subsumes more continuous regions of nodes 4 and 7, that are thus closed. Node inclusions are represented with dashed edges on Figure 2(c).

DAG:
The CHS produces a DAG. There exist multiple paths from the Root node to some other nodes. Note that the DAG structure allows disjoint diagnosis regions to be aggregated in the same node (see Figure 3).
3.4 Handling Continuous Variables

Computationally, one challenging aspect of the CHS is the handling of continuous variables. As previously mentioned, for \( n \), and \( \mathsf{H}(n) \) of cardinality \( k_n \), \( X_n \subseteq \mathbb{R}^{k_n} \). In algorithm 1, the expansion phase replicates the continuous state-space of a father node \( n \) into a child node \( n' \), such that \( X_n \times X_{n'} \subseteq \mathbb{R}^{k_n+1} \). In practice it is possible to maintain a single multidimensional space in \( \mathbb{R}^M \) where \( M \) is the total number of components in \( SD \). In this space, each conflict is a hypercube of dimension \( \leq M \). Step 2 of the CHS can be implemented as an intersection of all conflict hypercubes. This results into a partitioned hypercube of dimension \( M \).

Remaining operations translate into a labelling/unlabelling of the cube regions with the diagnoses of open nodes. In implementation we use bs-tries and the intersection operator in [Friedman et al., 1977].

3.5 Properties

Theorem 3 (Soundness of CHS). Let \( S \) be a set of conflict sets, and \( T \) a CHS-DAG obtained by using the CHS with node closing and pruning. For any open node \( n \) of \( T \), \( (\mathsf{H}(n), \Omega_n) \) is a minimal hitting set for \( S \).

Proof. Steps 4 and 8 ensure that any open \( (n, x) \) is a hitting set: it hits every conflict. It remains to show that \( (n, x) \) is minimal. Assume that \( (n, x) \) is not minimal. Then it exists an open node \((n', x')\) that is such that \( H(n') \subseteq H(n) \) and \( \Omega_{n'} \subseteq \Omega_n \), and that is not in \( T \). The CHS builds nodes from sets to supersets. Therefore \( n' \) must have been generated before \( n \) and must be in \( T \). So there is a \( n' \) such that \( H(n') \subseteq H(n) \). Thus either \( n \) or some other node \( n'' \) was expanded from \( n' \). This means there exist some \( x' \in X' \) such that they do not explain all conflicts in \( S \). But in this case, the \((n', x')\) are closed, either expanded or subsumed. Thus the \( x' \) are not in \( \Omega_{n'} \). So by contradiction of the assumption, \( (n, x) \) is minimal.

Theorem 4 (Completeness of CHS). Let \( S \) be a set of conflict sets, and \( T \) a CHS-DAG obtained by using the CHS with node closing and pruning. For any minimal hitting set \( (H^*, X^*) \), there exists an open node \( n \) of \( T \) such that \( (\mathsf{H}(n), \Omega_n) = (H^*, X^*) \).

Proof. Assume \( (H^*, X^*) \) minimal hitting set of size \( k \). Then there must be \( k \) components over \( S \) conflicts such that for every conflict \( s \in S \), \( s \cap H^* \neq \emptyset \). By construction of the DAG, for each conflict \( s \) CHS updates open nodes whose intersection with \( s \) is not empty, and expands all other open nodes \((n, x)\). So given the consideration above, there exists a path from the Root node to \( (H^*, X^*) \). This path goes through successively created nodes. However, all these nodes must be closed either: i/after being expanded into other nodes; ii/subsumed by some other nodes, which is impossible if \( (H^*, X^*) \) is minimal. So \( (H^*, X^*) \) marks the end of the path and is opened. In case \( k = 0 \), the Root node is the returned solution.

Alike the HS, CHS is incremental and takes conflicts in any order. Searching for all hitting sets of a given set is NP-complete, and the worst case performance of the standard HS is in the order of \( 2^M \). In fact, the observed performances are usually well under this theoretical bound but more realistic bounds of the HS performances are difficult to obtain. For the CHS, three cases can be distinguished, where in each conflict: i) each component has a single failure point; ii) each component has a single failure interval; iii) each component has disjoint failure intervals. The complexity of mixtures of these situations lies within the theoretical bounds for i), ii) and iii).

Assume \( M \) components over a set of \( K \) conflicts. The number of occurrence of component \( m \) over all conflicts is noted \( 0 \leq f_m \leq K \). In case ii), for \( m \), the maximum number of intervals over all conflicts is \( 2f_m - 1 \). This corresponds to the case where all intervals for \( m \) in conflicts do intersect with each others. Each intersected region thus explains a different subset of conflicts, and corresponds to different nodes of the CHS-DAG. Consequently an upper bound to the worst case performances is given by \[ \frac{1}{2} \sum_{m=1}^{M}[2f_m - 1] + \left( M \prod_{m=1}^{M} [2f_m - 1] \right) + \cdots + \prod_{m=1}^{M} [2f_m - 1]. \]

With it, bounds on cases i) and iii) can be easily expressed by considering unbounded intervals, and a fixed number of intervals per component, respectively.

4 Computational Improvements

In this section we propose two computational improvements to the CHS algorithm. One improves on the pruning computational efficiency, and the second yields a controlled approximation.

4.1 Efficient Implementation of the Pruning Loop

A dominant source of complexity of the CHS is the final closing of leaves (lines 13 to 16). The CHS creates new leaves for each conflict. In the worst case, there are \( md \) newly created leaves where \( d \) is the number of leaves before expansion and \( m \leq M \). Thus a naive implementation leads to a procedure that requires \( O(d^2m) \) inclusion checks (step 15). We develop a tree-based data structure that supports faster inclusion checks. The basic idea is to dissociate the inclusion check of the discrete element of diagnoses from the continu-
ous set inclusion check. By using an adaptation of the PATRI-
CIA tree (P-tree) [Morrison, 1968], the \((n, x)\) in newleaves
are grouped based on their discrete component \(n\) in a P-tree.

The newly created leaves of the CHS-DAG are located in the
leaves of the P-tree. The P-tree stores node \(n\) with an \(M\)-bit
key where bit \(i\) is positive if \(C_i \in H(n)\). Figure 10 depicts
such a tree for the newleaves list of example 1.

Insertion cost is \(O(M)\) bit checks for each new leaf. A
special operator performs an inclusion lookup for a node \(n'\)
that returns all \(n\) such that \(H(n') \subseteq H(n)\). For an \(M\)-bit
key with \(k\) negative bits and inclusion lookup table 10(b), an
inclusion lookup costs \(2^k\) positive bit checks, and \(M - k\)
negative bit checks per explored tree branch, so total cost is
\(O(2^k(M - k^3))\) bit checks. Clearly, the advantage is that
with this storage structure, the inclusion lookup does not de-
pend on \(d\) or the number of conflicts. Thus in practice we ob-
serve a good speed-up of the pruning loop that improves the
CHS scalability. See the result section for empirical details.
The inclusion test is performed for each of the oldleaves ele-
ments. The continuous inclusion check is performed on those
new leaves that are returned by each inclusion lookup.

4.2 Generating Relevant Diagnoses

In Artificial Intelligence it is important to study the genera-
tion of approximated results. Here the idea is developed that
some nodes of the CHS DAG are more important than others.
Suppose that each component \(C\) has a probability distribution
\(p(AB(C)|x)\) of failing over \(x\). The probability of a node
\((n, x)\) is:

\[
p_n(x) = \prod_{C_i \in H(n)} p(AB(C_i)|x) \tag{4}
\]

and the probability of \(n\) is:

\[
p_n = \int_{\Omega_n} p_n(x)dx. \tag{5}
\]

The CHS algorithm can be easily adapted to the computation
of the most relevant minimal diagnoses. Given a number \(\epsilon\)
between 0 and 1, the nodes \((n, \Omega_n)\) with \(p_n < \epsilon\) are closed.

5 Experimental Evaluation

The CHS was implemented and tested extensively through simula-
tion experiments. Overall, it yields fast results for spaces under 10 dimensions, but doesn’t scale favorably well
beyond. The main results are drawn from a set of 500 runs of
the CHS for \(M = K = 6\) components and conflicts. The
simulation settings were as follows. Each conflict has a ran-
dom size. Each component in a conflict comes with a random
interval that is generated by picking up two integers between
0 and 100.

This section reports on the reactions of the CHS. The con-
tinuous diagnoses are the open continuous leaves of the CHS-
DAG and their number is the total number of minimal di-
agnoses. The discrete diagnoses are the open nodes of the
CHS-DAG. Both numbers theoretically grow exponentially
with the total number of conflict elements. This is visible on
Figure 4 despite the fact that our experiments were limited to
small numbers of components. In consequence the CHS has
expanded many of the discrete diagnoses after just a few conflicts (Figure 5). The DAG structure in the aggregate space of diagnoses allows the minimal continuous diagnoses to continue to grow after all discrete diagnoses have been expanded (Figure 4).

The complexity analysis has shown how the number of occurrence of a component in conflicts plays a crucial role. This is clearly confirmed on Figure 6. The explosion of the number of minimal continuous diagnoses is a direct consequence of the NP-complete nature of the problem. Figure 7 shows the minimal discrete diagnoses are distributed differently. This is due to the DAG structure: given a mean integer \( f \) of mean occurrences over \( M \) components, this number is always smaller than \( \sum_{i=1}^{f} \binom{M}{i} \). That is, the number of minimal discrete diagnoses is at most all combinations of \( f \) and fewer components.

Based on a second set of experiments we aimed to elucidate the scaling properties of the approach wrt. the continuous dimensions. These experiments are runs with \( M \) ranging from 1 to 11, \( K = 4 \), and conflict random intervals in \([0, 10]\). The results are graphically depicted on figures 8 and 9. The exponential response of the number of minimal continuous diagnoses appears clearly.

![Figure 11: Expansion vs. pruning.](image)

In practice it was not possible to run the CHS in reasonable time on problems with more than 10 or so continuous dimensions. The limitation stems mainly from the pruning phase that dominates the computational effort (Figure 11). In worst cases however, the pruning loop continues to require up to several billions inclusion checks of continuous sets.

6 Conclusion

We have presented the CHS algorithm, a solution to finding the minimal hitting sets of a collection of finite sets whose elements have continuous extents. The algorithm uses an DAG representation in an aggregate space of diagnoses. CHS is based on the same dual mechanism as the classical hitting set algorithms: it has an expansion phase and a pruning phase. To our knowledge CHS is the first computational method to produce minimal diagnoses conditioned over continuous regions. In practice however, CHS exhibits an unfair behavior: it expands high numbers of potential diagnoses in little time and spends most of its time pruning out a large fraction of them. It is an open problem how to better tackle this computational cost.

References


Optimal observability for diagnosability

Laura Brandán Briones∗, Alexander Lazovik++ and Philippe Dague∗∗
LRI, Univ.Paris-Sud, CNRS and INRIA Saclay - Île-de-France
Parc Club Orsay Université, Orsay, F-91893, France.
{laura.brandan,alexander.lazovik,philippe.dague}@lri.fr

Abstract
The diagnosability property recognizes if a system model can be unambiguously diagnosable; that is, if all faults can be detected using only the information given by the observable events. Usually, large and complex systems require an automatic fault detection and isolation, but to specify the minimal observability degree of a system to be diagnosable is not a trivial task.

In this paper we give the necessary and sufficient conditions for observability (the list of observable events) that a system has to maintain to be diagnosable. We concentrate on two problems: first, we transform a diagnosable system into one with minimal degree of observability and still diagnosable. Second, we transform a non-diagnosable system into diagnosable by increasing the degree of observability. We also expand the developed algorithms with several extensions: for distinguishability, for predictability and for extended fault models.

1 Introduction
Fault detection is an extremely important task, and its automatization has been studied for several years. The increasing reliability requirements on autonomous systems, especially mission-critical ones, have resulted in the development of sophisticated methods for the accurate analysis of faults. One of the most important reliability properties in large systems is their diagnosability. The diagnosability property recognizes if a system model can be unambiguously diagnosable, that is, if any occurred fault is detectable within predefined finite number of steps. This recognition is performed by observing the system, e.g., by receiving information from sensors, or by monitoring logs of the running software.

However, when a system is being set up, there is always the problem of how much observability is necessary to keep the system diagnosable? In real world this question may sound as “how many sensors should be installed?” or “how many events should be monitored?”. In particular, for non-diagnosable legacy systems this question may sound as “How many new sensors or monitors should be installed, to make the system diagnosable?”. More subtly, in diagnosable legacy systems: “Are there too many sensors or monitors, making unnecessary and redundant the collected information?”.

This paper gives the most efficient system observability (list of observable events) that keeps the system diagnosable. Following Lin work [8] where it is proved that the most efficient system observability is not unique, we extend it in two ways: First, we give an algorithm that keeps a system diagnosable but with a minimal set of observable events. Second, we present an algorithm that builds the optimal set of observable events to ensure diagnosability in a given system.

Often, especially in legacy systems, one cannot modify anything in the running system but the level and intensity of the observation (e.g., sensors or logs of the system). In this paper we assume that we are only permitted to transform the degree of observability of the system, but not the system itself.

To deal with faults in a uniform way, we introduce a signature definition that describes the situations when a fault occurs. The notion of signature is a very well known concept in Continuous Systems (CS). There the diagnosis is often performed on a snapshot of observables, i.e., an evaluation of observable variables at a given point of time. If the observation implies a fault occurrence then the observation is called a signature. However, in discrete event systems (DES) approaches the diagnosis reasoning is typically more dynamic, i.e., with different observations for a period of time. Therefore, the signature concept has to be re-defined.

In this paper we propose a definition of signatures for DES, which allows us to deal with different fault situations in a uniform way. This definition provides us with two advantages:
1. An efficient method for reducing and expanding the observability of a given system.
2. An extra layer between the actual fault model and the algorithms, which makes possible to develop algorithms independent from the actual nature of the faults.

With signatures, we build a framework that is sufficiently general for our purpose of reducing and expanding observability, and easily extendable to a wide range of problems.

The main contribution of this paper is the analysis of dif-
Different observability levels for DES. We concentrate on two approaches: first, we transform a diagnosable system into one with minimal observability but still diagnosable. Second, we transform a non-diagnosable system into a diagnosable one by increasing the system observability. We illustrate our propositions with an intuitive example throughout the paper.

Our algorithms work and transform only the observability (the list of observable events) of the system without changing its structure (set of transitions, events or states), which makes them ideal to apply in situations where the system is fixed, and the only change available is the level of system monitoring (e.g., via installation of additional sensors).

This paper also presents extensions to the problem of reducing and expanding observability, namely: (i) distinguishability, where it is important to differentiate the fault types; (ii) predictability, where the fault is predicted rather than detected a posteriori; (iii) extended fault model, where a fault is formed by a specific faulty sequence of events, that are not faults by themselves.

**Organization of the paper** Section 2 overviews the related work. Section 3 provides basic definitions, e.g., label transition system, diagnosability. In Section 4 we introduce the notion of signatures and correct behaviours. The framework for reducing and expanding observability are presented in Section 5, with extensions provided in Section 6. Finally, we draw the conclusions in Section 7.

## 2 Related work

Diagnosability study for discrete-event systems is not new. In [12] diagnosability is precisely defined and an algorithm for checking diagnosability is provided. In this paper we go beyond diagnosability checking and discuss different levels of observability for diagnosable discrete-event systems. Assuming that we have a diagnosable system we develop an algorithm to find a minimal set of observable events that still keeps the system diagnosable. We also provide an algorithm that transforms non-diagnosable system into diagnosable one by increasing the system observability. There are different ways of representing the partial observability in DES (see, for instance [9]), although, there was only limited attention to how the system observability level affects its diagnosability, e.g., [6]. In contrast to [6], we introduce the notion of a signature to abstract from the underlying failure model. It originally comes from continuous systems [13]. In this work we adopt signatures to represent fault executions for DES. It [15] is proven that finding the optimal (minimal) set of sensors (in our case observable events) for a diagnosable system is NP-complete.

Diagnosability problem is somewhat related to the problem of model checking. However, there is an important difference. Model checking algorithms verify if the (possibly infinite) executions of a system satisfy a given property. Merely, it checks if there is a fault in the system, while diagnosability verifies if the existing faults are detectable. However, there is some work, that shows how diagnosability can be represented as a model checking problem [1].

The approach of extending the fault model presented in Section 6 is not new. For example, in [5] faults are described as formulae in linear temporal logic, in [4] a notion of a supervision pattern is introduced to allow more complicated pattern-based models of failures.

Our approach makes no assumption on whether the system has control over its events, making the system being “passively” diagnosable. In contrast, in active diagnosis [11], the controller is designed to take into account the issues of diagnosability. Similar problems are also tackled in planning, where a planner can decide on the most appropriate actions to deal with faults, see for instance [7].

In this paper our work is mostly oriented to the definition of the general theoretical framework, and does not address the problems related to practical application of the proposed techniques. However, as part of the future work, we plan to evaluate the developed framework against real cases [14].

In [8] to diagnose a fault is to be able to identify in which state or set of states the system is. In contrast, we do not require information about states, we require only to be certain that a fault has occurred.

Our notions of predictability are closely related to the work in [2; 3], where the problem of predicting occurrences of a fault is addressed. We extend these works with definitions of safe predictability and strong predictability. Furthermore, we define the signature concept for these two definitions and show how our framework can be applied to predictability.

## 3 Discrete Event Systems

### 3.1 Preliminaries

Let $L$ be any set. Then with $L^*$ we denote the set of all finite sequences over $L$, with $L^\omega$ we denote the set of all infinite sequences over $L$ and with $L^n$ we denote the set of all finite and infinite sequences over $L$. The empty sequence is denoted by $\varepsilon$. We use $L^n = L^n \setminus \{\varepsilon\}$. For $\sigma, \rho \in L^n$, we say that $\sigma$ is a prefix of $\rho$ and write $\sigma \subseteq \rho$, if $\rho = \sigma\sigma'$ for some $\sigma' \in L^n$ (then $\sigma' = \rho - \sigma$). If $\sigma$ is a prefix of $\rho$, then $\rho$ is a continuation of $\sigma$. We call $\sigma$ a proper prefix of $\rho$ and $\rho$ a proper continuation of $\sigma$ if $\sigma \subset \rho$, but $\sigma \neq \rho$. We denote by $P(L)$ the power set of $L$. Given $L' \subseteq L$ and a sequence over $L^\omega$ we denote by $\sigma_{L'}$ the restriction of $\sigma$ over $L'$.

### 3.2 Labelled transition systems

**Definition 1 (LTS).** A labelled transition system, LTS, is a tuple $A = \langle Q, q^0, L, T \rangle$ where $Q$ is a finite set of states; $q^0 \in Q$ is the initial state; $L$ is a finite set of events; $T \subseteq Q \times L \times Q$ is the finite branching transition relation. We denote the components of $A$ by $Q_A$, $q^0_A$, $L_A$, and $T_A$. We omit the subscript $A$ if it is clear from the context.

In Figure 1(A) we represent $A = \langle Q, q^0, L, T \rangle$ a LTS where $Q = \{q_0, \ldots, q_8\}$, $q^0 = q_0$, $L = \{a, b, c, d, e, f, i, j\}$ and $T = \{(q_0, b, q_1), (q_0, a, q_2), \ldots, (q_8, a, q_8)\}$.

**Definition 2 (path, trace, $\sigma$, $q \xrightarrow{\sigma} q'$, cycle, $\delta^\sigma$).** Let $A = \langle Q, q^0, L, T \rangle$ be a LTS, then $A$ is a path in $A$ is a sequence $\sigma = q_0q_1q_2 \ldots$ such that for all $i$ we have $(q_i, a_{i+1}) \in T$. We denote with $\text{paths}(q)$ the set of paths starting in $q$. We use $\text{paths}(A)$ for paths($q^0$). We denote with $\text{paths}(q, q')$ the set of paths starting in $q$ and ending in $q'$. We write $q \rightarrow q'$, if $\text{paths}(q, q')$ is not empty and $q \rightarrow$, if there exists a state $q'$ such that $q \rightarrow q'$.
– The trace \( \sigma \) of a path \( \pi \), denoted \( \text{trace}(\pi) \), is the sequence \( \sigma = a_0a_1 \ldots \) of events in \( L \) occurring in \( \pi \). We write \( \text{traces}(A) = \{ \text{trace}(\pi) \mid \pi \in \text{paths}(A) \} \) for the set of traces in \( A \), particularly we write \( \text{traces}^{\infty}(A) \) to denote the set of infinite traces in \( A \). In case \( \sigma \) is finite, with \( |\sigma| \) we denote the length of the trace \( \sigma \) and we define by \( \text{last}(\sigma) \) the last event of \( \sigma \).

– We write \( q \xrightarrow{a} q' \) if the state \( q' \) can be reached from the state \( q \) via the trace \( \sigma \), i.e., if there is a path \( \pi \in \text{paths}(q,q') \) such that \( \text{trace}(\pi) = \sigma \).

– A cycle is a non-empty element in \( \text{paths}(q,q) \) for some state \( q \). We denote by \( \text{cycle}(A) \) all the cycles in \( A \).

– Given a trace \( \sigma \in \text{traces}(A) \), we denote by \( \bar{\sigma} \) its postlanguage, i.e., \( \bar{\sigma} = \{ \rho \in \text{traces}(A) \mid \sigma \sqsubseteq \rho \} \). Moreover, for a given natural number \( k \in \mathbb{N} \) we denote by \( \bar{\sigma}^k \) its postlanguage with words of length equal or longer than \( |\sigma| + k \), i.e., \( \bar{\sigma}^k = \{ \rho \in \bar{\sigma} \mid k \leq |\rho| - |\sigma| \} \).

We say that a LTS \( A \) is live if for all states there exists a transition initiated in that state, i.e.,
\[
\forall q \in Q : q \rightarrow (1)
\]

For example, the LTSs from Figure 1-(A) (on the left) and 1-(B) (on the right) are live.

### 3.3 Observable LTSs with faults

An observable labelled transition system with faults is a LTS that has as its set of events subdivided into observable events \( (L_o) \) and unobservable events \( (L_u) \). Moreover, there exists a subset of \( L_o \) that represents fault events \( (L_f) \).

**Definition 3 (observable LTS \( (L_f) \))**. An observable labelled transition system with faults, denoted LTS \( (L_f) \), is a tuple \( A_{(L_f)} = (Q, q^0, L, T, L_o, L_u, L_f) \) where \( (Q, q^0, L, T) \) is a LTS and:

- The set of events \( L \) is partitioned into a set of observable events, \( L_o \), and a set of unobservable events, \( L_u \), with \( L = L_o \cup L_u \) and \( L_o \cap L_u = \emptyset \).
- There is a subset of the unobservable events, called the fault events, denoted \( L_f \).

In a sense, an observable LTS \( (L_f) \) is about hiding the faults and some other events. From now on, we refer to LTS \( (L_f) \) as to observable LTS \( (L_f) \) unless the state the opposite.

As an example, Figure 1-(A) represents a LTS \( (L_f) \) with:
\[
L_u = \{ f_i, f_j \} \text{ and } L_f = \{ f_i, f_j \}.
\]

**Definition 4 (observable trace, \( \text{traces}^{\infty}, \text{traces}^{f \downarrow} \)).** Let \( A_{(L_f)} = (Q, q^0, L, T, L_o, L_u, L_f) \) be a LTS \( (L_f) \), then:

- The observable trace of a trace \( \sigma \), denoted \( \sigma_{L_o} \), is the sequence \( a_0a_1 \ldots \) of events in \( L_o \) occurring in \( \sigma \).

- We denote by \( \text{traces}^{f \downarrow}(A) \) the set of traces in \( A \) that end with a fault, i.e., \( \text{traces}^{f \downarrow}(A) = \{ \sigma \mid \sigma \in \text{traces}(A) \} \).

- Given a natural number \( k \in \mathbb{N} \) we denote by \( \text{traces}^{f \downarrow k}(A) \), the set of traces \( \sigma \) such that there exists another trace \( \sigma' \) that ends in a fault and \( \sigma' \) extends \( \sigma' \) with \(|\sigma'| + k \) or more events, i.e., \( \text{traces}^{f \downarrow k}(A) = \{ \sigma \in \text{traces}(A) \mid \exists \sigma' \in \text{traces}^{f \downarrow}(A) \land \sigma' \in \sigma \} \).

We say that a LTS \( (L_f) \) is diagnosable if it does not have cycles with non-observable events, i.e.,
\[
A \text{ is diagnosable, if and only if, } \forall \pi \in \text{cycle}(A) \exists a \in L_o : a \in \pi
\]
necessarily non diagnosable. A more interesting situation is when a system is diagnosable with faults being the only non-observable events. We denote such systems possibly diagnosable. To distinguish if there exists an observability degree, for a particular system, that makes it diagnosable, we propose thus the following definition.

**Definition 6 (necessarily non/possibly diagnosable).** A LTS($L_f$) A is called:

- Necessarily non diagnosable if the LTS($L_f$) $A(L_f)$ is not diagnosable.
- Possibly diagnosable if the LTS($L_f$) $A(L_f)$ is diagnosable.

From now, we assume that all LTS($L_f$) $A$, that we work with, are live, convergent and possibly diagnosable.

### 4.2 Signatures in DES

In this section we introduce the notion of signatures that originally comes from continuous systems [13]. In this work we adopt signatures to represent faults executions for discrete-event systems. A signature is a regular expression that denotes the set of traces with faults. The observable correct behaviour, denoted $e$, is defined by the regular expression (with only observable events) that denotes all correct infinite traces, i.e., infinite traces that do not have faults.

**Definition 7 (observable correct behaviour).** Given a LTS($L_f$) $A$ we define its observable correct behaviour as:

$e = \{ \sigma \in L_o^\infty \mid \sigma \in \text{traces}^\infty(A) : f \not\in \sigma \}$

In contrast with the observable correct behaviour, the observable signature is the regular expression that denotes all observable prefixes of faulty traces that are not prefix of an element of the observable correct behaviour.

To define observable signatures we use $\text{traces}^\infty$ to denote the set of all traces with exactly $z$ events after the first occurrence of a fault. In [10] it is shown, that for an exhaustive diagnosability check, it is necessary to check traces of a maximal length $\frac{t-1}{2}$ (and, therefore we can choose $z \leq \frac{t-1}{2}$), what is often impractically high. In practice, to build a correct signature, one is sometimes forced to set an upper bound for the length of faulty traces.

**Definition 8 (observable signature).** Given a diagnosable LTS($L_f$) $A = (Q, q_0, L, T, L_o, L_f)$, its observable signature is defined as:

$r = \{ \sigma \in L_o^\infty \mid \exists \alpha \in \text{traces}^\infty(A) : \sigma \subseteq L_o \wedge \exists \sigma' \in c : \sigma \subseteq \sigma' \}$

Note that the observable correct behaviour is not the complement of the observable signature: there exist traces that do not belong to either class.

**Remark 3:** A consequence of the diagnosability definition (Definition 5) is that $A$ being diagnosable ensures that $r$ is empty (from the moment $A$ contains at least one fault) with a $z = \frac{t-1}{2}$.

Following the example from Figure 1-(A), with $L_o = \{f_1, f_2\}$ we can find $c = abda^\infty + bbd^a\infty \, z = 5$, then $\text{traces}^\infty = \{afbea^3, bfbca^3\}$. So, finally we can obtain the observable signature $r = abe + abea + abea^2 + bEase + bbea + bbea^2 + bbea^3$. In particular, we let observable signatures represent traces that do not have faults, but are prefixes of faulty traces. In Figure 1-(B) we represent a diagnosable system that is bound to have a fault with $L_o = \{f\}$. We obtain $c = \emptyset$ because there is no infinite trace without fault. In addition, $\text{traces}^\infty = \{af, baa\}$ then we can obtain $r$ as $a + ab + aba + aba^2 + aba^3$.

### 5 Reducing and expanding observability

#### 5.1 Reducing the observability

In this section we find a minimal set of observable events that still keeps the system diagnosable. We do it by reducing the set of observable events as much as possible still maintaining the diagnosability property.

The observable signatures defined in Definition 8 describe the observable part of traces with faults or traces that certainly will produce a fault. Although, for the following proofs we use a more restricted version of signature, called long signature and denoted $lr$. Long signatures do not contain traces that are prefix of another one, i.e.,

$$lr = \{ \sigma \in L_o^\infty \mid \sigma \in r \wedge \exists \sigma' \in r : \sigma \subseteq \sigma' \}$$

For example, our previous signature $r = abe + abea + abea^2 + bbea + bbea^3 + bbea^2 + bbea^3$. Moreover, for a trace $a$ we abuse the notation: $a \in r$, $a \in lr$ and $a \in c$ to denote that the observable trace of $a$ (i.e., $la$) is in $r$ or $c$ respectively.

**Structural differences,** written $A \not\equiv B$, relate two sets of traces that do not have any trace in common, nor any prefix of a trace that belongs to the other set, except of the $e$ trace. Formally, structural differences are defined as follows.

**Definition 9 (structural differences ($\not\equiv$)).** Let $A$ and $B$ be sets of traces, then $A \not\equiv B$ means $(\forall \sigma \in A : \exists \sigma' : \sigma' \not\in B) \wedge (\forall \sigma \in B : \exists \sigma' : \sigma' \not\in A)$

**Lemma 1.** Let $A(la) = \{Q, q_0, L, T, L_o, L_f\}$ be a diagnosable LTS($L_f$), then if $a \in L_o$: $lr_{la\{a\}} \not\equiv c_{la\{a\}}$ and $A(la) = \{Q, q_0, L, T, L_o, L_f\}$, with $L_o = L_o \cup \{a\}$ is convergent then $A(la)$ is diagnosable, and not diagnosable otherwise.

**Proof.** Let $lr' = lr_{la\{a\}}$ and $c' = c_{la\{a\}}$. Note that for every $\sigma \in lr$ and $c \in c'$ it follows that $\sigma \in lr'$ and $\alpha \in c'$. To prove the first part, we have to show that $\exists n: \forall |\sigma'| > n : \sigma \equiv \sigma'$, that $\sigma' \equiv \sigma'$, where $L_o = L_o \cup \{a\}$. Let $n = z = \frac{t-1}{2}$. From $\forall \sigma \in \text{traces}^\infty(A(\lambda_a)) : \sigma \equiv \sigma'$, it follows $\sigma \in lr \Rightarrow \sigma \in lr'$. Suppose, $\exists \sigma_1 : f \not\in \sigma_1 \wedge \sigma_2 \equiv \sigma_1$. Because $\alpha \equiv \sigma_1$ and $\alpha \in lr'$ then $\sigma_1 \in lr \Rightarrow \sigma_1 \in lr'$. There are two possible cases: 1. $\exists \sigma_2 \in \text{traces}^\infty : \sigma_2 \equiv \sigma_2 \wedge f \not\in \sigma_2$. Then $\sigma_2 \in c'$. Because $lr_{l_a} \not\equiv \sigma_2$, then $\sigma_2 \in lr'$. But then $\sigma \not\in lr'$ which contradicts with $\sigma \in lr'$. 2. $\exists \sigma_2 \in \text{traces}^\infty : \sigma_2 \equiv \sigma_2 \wedge f \not\in \sigma_2$. That also means that $\exists \sigma_3 \not\in \text{traces}^\infty : \sigma_3 \equiv \sigma_3 \wedge f \not\in \sigma_3$. From Property 1 it follows then that the system is diagnosable.

Let us now prove, that if the assumption $lr_{l_a} \not\equiv \sigma_2$, does not hold, then the system becomes not diagnosable if $a$ is removed from $L_o$. If the assumption does not hold, then $\exists \sigma \in lr' : \exists \sigma_1 \in c : \sigma \subseteq \sigma_1$. And from it follows either
Algorithm 1 Reducing the observability

1: Input: System $A$, observable events $L_o$, observable correct behavior $c$, observable signature $r$
2: Output: Minimal set of observable events $L'_o$
3: $lr = \text{reduceToLongSignature}(r)$
4: $S_{max} = \emptyset$
5: for all $S \subseteq L_o$ do
6: if checkUnObSet$(A, S, c, lr)$ and $|S_{max}| < |S|$ then
7: $S_{max} = S$
8: return $L_o \setminus S_{max}$
9: function checkUnObSet$(A, S, c, lr)$
10: with $a \in S$ do
11: if isConvergent$(A, L_o \setminus a) \land$
12: checkUnObserve$(c, lr, a)$ then
13: return checkUnObSet$(A, S \setminus \{a, c \setminus \{lr - \{a\}\})$
14: function checkUnObserve$(c, lr, a)$
15: for all $\sigma \in (c \setminus \{a\})$ do
16: for all $\sigma' \in \text{prefix}(\sigma) \land (|\sigma'| \leq |lr|)$ do
17: if $\sigma' \leq (lr - \{a\})$ then
18: return false
19: return true

The above procedure can be performed according to different orders, depending on which observable events we choose to turn into unobservable ones. Note that there always exists a minimal order w.r.t. the amount of observable events. The algorithm itself for reducing the observability is shown in Algorithm 1. It provides an algorithmic view for Lemma 1 and Theorem 1. Given a system, its observable events and observable signature, the algorithm returns a minimal set of observable events. It works in the following way. In line 3, it reduces the observable signature according with (3). In line 5-7, the algorithm chooses the set $S$ with maximal cardinality, which is built by functions checkUnObSet and checkUnObserve. The functions form the set $S$ by iteratively reducing the set of observable events as far as observable signature for $lr$ and observable correct behavior $c$ are still distinguishable, in the same way as it was defined by Lemma 1.

In the following example we show how we obtain the minimal set of observable events for the system $A$ from Figure 1-(A). Starting from $A(L_o) = (Q, q^0, L, T, L_o, L_f)$ with $L_o = \{f_1, f_2\}, L_o = \{a, b, c, d, e\}$, we get $lr = aab + abe + abea + abc + bba + bcb + bca + bba$ and $c = ada + bda$. Then, (i) we obtain $lr = aab + abe + abc + bba + bba$; (ii) we convert $c$ into unobservable, having $lr = aab + abe + abc + bba + bca + bbb$; (iii) we convert $b$ into unobservable, having $lr = aab + abe + abc + bba + bbb$; and $c = ada + bda + bda + bda$; (iv) we convert $c$ into unobservable, having $lr = aab + abe + abc + bba + bbb$; and (v) we reconstruct the signature as $r = aa + aad + aad$. It is easy to note that $A(L'_o) = (Q, q^0, L, T, L_o, L_f)$ with $L_o = \{f_1, f_2, b, e, c\}$, $L_o = \{a, d\}$ is diagnosable.

5.2 Expanding the observability

In this section we present the algorithm to transform a non-diagnosable system into a diagnosable one, expanding its set of observable events.

We assume that the system is possibly diagnosable (Definition 6). Thus, if we consider the system with all events, except faults, as observable, then the system is diagnosable. We define $S_{\sigma, \alpha}$ as a set of sets of events that distinguish traces $\sigma$ and $\alpha$. So, in a possibly diagnosable system with two traces $\sigma$ and $\alpha$ with the same observability such that one has a fault, and afterwards it has at least $n$ events, and the latter one without a fault, we define $S_{\sigma, \alpha}$ as a set of sets of events (not from $L_f$) that makes $\sigma$ and $\alpha$ distinguishable.

Definition 10 ($S_{\sigma, \alpha}$). Let $A(L_o)$ be a possibly diagnosable LTS $(L_f)$, then $\forall \sigma, \alpha \in \text{traces}(A) : \sigma \in \text{traces}(A) \land \alpha \notin \text{traces}(A)$, we define

$$S_{\sigma, \alpha} = \{Q \subseteq L_o \mid \sigma_{\text{loc}} \neq \alpha_{\text{loc}}, \sigma\}$$

where $n$ is the bound, given by the diagnosability definition, for the system $A(L_o)$.

Considering the system presented in Figure 1-(A), with $L_o = \{a, b\}, L_o = \{c, d, e, f_1, f_2\}$, $\sigma = bfbc$ and $\alpha = bbd$ (so $\sigma_{L_o} = bb$ and $\alpha_{L_o} = bb$) we have $S_{\sigma, \alpha} = \{\{d\}, \{c\}, \{e\}\}$. Now, with $\sigma' = afbc$ and $\alpha' = abd$, we obtain $S_{\sigma', \alpha'} = \{\{e\}, \{d, e\}, \{d\}\}$.

A minimal distinguishable set, denoted by $S$, represents a set that includes at least one set for all $S_{\sigma, \alpha}$.

Definition 11 ($S$). $S$ is a minimal distinguishable set, if it has minimal cardinality and for all $S_{\sigma, \alpha}$ with
Algorithm 2 Expanding the observability

1: Input: System $A$, observable events $L_o$
2: Output: Minimal set of observable events $L'_o$
3: for all $\sigma \in \text{traces}(A) \land |\sigma| \leq z$ do
4:   for all $\alpha \in \text{traces}(A)$ : $\alpha|_\sigma \neq \sigma|_\sigma \land f \neq \alpha$ do
5:      $S_{\alpha \sigma} = \{O | \sigma|_{L_o \setminus \sigma} \neq \sigma|_{L_o \setminus \sigma} \land f \notin O\}$
6:      $S = \emptyset$
7:   for all $S_{\alpha \sigma}$ do
8:      $S = (B' \cup B'' \mid (B', B'') \in S \times S_{\alpha \sigma})$
9:      for all $B, B' \in S, B' \subseteq \sigma$ do
10:    $S = S - B$
11: return $L_o \cup \text{B}_{\sigma \alpha}$, where $\text{B}_{\sigma \alpha} \subseteq S \land |\text{B}_{\sigma \alpha}| = \min |B', B \in S$

For the example shown in Figure 1-(A) we have $S = \{d\}$.

Theorem 2. Let $A(L_o) = \langle Q, q^0, L, T, L_f, L'_o \rangle$ be a possible diagnosable but non-diagnosable LTS($L_f$), and let $S$ defined as in Definition 11, then $A(L'_o) = \langle Q, q^0, L, T, L'_o, L'_f \rangle$ is diagnosable, with $L'_o \subseteq L_o \setminus S$.

Proof. Suppose $A(L'_o)$ is non-diagnosable. Therefore there exist $\sigma, \alpha \in \text{traces}(A)$ such that $\sigma \in \text{traces}^{\sigma, \alpha}(A) \land f \notin \alpha : \sigma|_{L_o} = \alpha|_{L_o}$, and then, using Definition 10 there exist $S_{\sigma \alpha}$ and, moreover, from Definition 11 there exist $B \in S_{\sigma \alpha} : B \subseteq S$, so $\sigma|_{L_o} \neq \alpha|_{L_o}$.

We have a contradiction, that comes from supposing that $A(L'_o)$ is non-diagnosable.

The algorithm is provided in Algorithm 2. It transforms the system into a diagnosable one without changing its structure (set of transitions, events or states) and only by expanding its observability. Moreover, the algorithm finds a minimal set of events that should be added to the initial set of observable events. The algorithm itself is based on the definitions and the theorem above: lines 3-5 refer to Definition 10, lines 6-10 refer to Definition 11 and Theorem 2. Note, that in line 3 we use $z$ to limit the maximum length of $\sigma$. For an exhaustive search $z$ has to be equal to $2^m - 1$. In practice it is often possible to provide a better bound (e.g., [10]). Note that, depending on the initial $L_o$, $L'_o$ is not necessarily a minimal set of observable events making the system diagnosable. However, the minimality can always be reached by applying the algorithm presented in Section 5.1 from $L_o = L'_o$.

In our example from Figure 1-(A) with $L_o = \{a, b\}$, following Algorithm 2 we obtain $S = \{d\}$. In contrast with the minimal set of observable events for this diagnosable system that is $L_{o_{\min}} = \{a, d\}$, as we already pointed out in Section 5.1, we obtain $A(L'_o) = \langle Q, q^0, L, T, L'_o, L'_f \rangle$ with $L'_o = \{a, b, d\}$.

6 Extended models

In this section we introduce various extensions to the diagnosability model that we presented in Section 3.4. Within the framework defined in Section 4, it is possible to reuse the algorithms from Theorem 1 and Theorem 2, and, in the same time, take into account several extensions: distinguishability, predictability, and extended fault models. In the following subsections, we show what has to be modified in the proposed model and algorithms to deal with each particular case.

6.1 Distinguishability

The problem of distinguishability arises when we are interested in distinguishing different types of faults rather than in a simple indication whether a fault occurred or not.

In this section we partition the set of faults (subset of the unobservable events) into classes of faults, i.e., $\Pi_f = \{L_{f_1}, \ldots, L_{f_k}\}$, where $L_{f_i}$ represents all faults with type $f_i$.

Definition 12 (observable LTS($\Pi_f$)). An observable labelled transition system LTS($\Pi_f$) with fault types, denoted by $\Pi_f$, $A(L_o) = \langle Q, q^0, L, T, L_f, L'_o \rangle$ is a LTS($L_f$), where the set of fault events ($L_f$) is partitioned into $\Pi_f = \{L_{f_1}, \ldots, L_{f_k}\}$, i.e., $L_f \subseteq L_{f_1} \cup \cdots \cup L_{f_k}$ and $\forall i \neq j : L_{f_i} \cap L_{f_j} = \emptyset$.

So an observable LTS($\Pi_f$) is a normal LTS with a clear distinction between observable and unobservable events and inside the unobservable events there is a subset of fault events subdivided into classes.

Definition 13 (traces, traces$^{k_f}$, $f \in \sigma$). Let $A = \langle Q, q^0, L, T, L_f, L'_o \rangle$ be a LTS($\Pi_f$), then:

- Given a type of fault $f_i$ we denote by traces$^{f_i}$($A$) the set of traces in $A$ that end with a fault of type $f_i$ i.e., traces$^{f_i}$($A$) = $\{ \sigma \in \text{traces}(A) | \sigma \in L^*_f \}$.
- Given a type of fault $f_i$ and a natural number $k \in \mathbb{N}$ we denote by traces$^{f_i, k}$($A$), the set of traces $\sigma$ such that there exists another trace $\sigma'$ that ends in a fault of type $f_i$ and $\sigma$ extends $\sigma'$ with length longer or equal to the length of $\sigma'$ plus $k$, i.e., traces$^{f_i, k}$($A$) = $\{ \sigma \in \text{traces}(A) | \exists \sigma' \in \text{traces}^k(A) \land \sigma \subseteq \sigma' \}$.
- Given a trace $\sigma$, we write $f_i \in \sigma$ to denote that $\sigma$ has a fault of type $f_i$, i.e., $\sigma \in L^*_f L^*_f$.

As follows, we re-define diagnosability and observable signatures for LTS($\Pi_f$).

Definition 14 (diagnosability in LTS($\Pi_f$)). Let $A(L_o) = \langle Q, q^0, L, T, L_f, L'_o \rangle$ be a LTS($\Pi_f$), then the set traces($A$) is diagnosable if the following holds,

\[
\forall 1 \leq i \leq m : \exists n_i \in \mathbb{N} : \forall \rho \in \text{traces}^{f_i, n_i}(A) \land \alpha \in \text{traces}(A) \land \rho|_{\alpha} = \alpha|_{\alpha}, then \exists f_i \in \alpha.
\]

Definition 15 ($r'$). Given a diagnosable LTS($\Pi_f$) $A$ and $f_i$ a fault type; $r'$ is the observable signature of $f_i$ if it observable prefixes of traces contain a fault of type $f_i$ that are not prefix of a correct trace.

\[
r' = \{ \sigma \in L^*_o | \exists \alpha \in \text{traces}^{f_i, n_i}(A) : \sigma \subseteq \alpha|_{L_o} \land \sigma \neq \alpha|_{L_o} \}.
\]

Definition 16 (signature in LTS($\Pi_f$)). Given a LTS($\Pi_f$) $A$ and $r_1, \ldots, r^n$ the set of observable signatures for fault types $f_1 \cdots f_m$, we define the observable signature of $A$: $r = r_1 + \cdots + r^n$.

Now we can reformulated Lemma 1 w.r.t. different faults.

Lemma 2. Let $A(L_o) = \langle Q, q^0, L, T, L_f, L'_o \rangle$ be a diagnosable LTS($\Pi_f$), then if $a \in L_o : r|_{\alpha|_a} \neq c|_{\alpha|_a}$ and $\forall i \neq j : r|_{L_o \setminus \{a\}} \neq r|_{L_o \setminus \{a\}}$ then $A(L_o) =$

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
\( \langle Q, q^0, L, T, L' \rangle \), with \( L'_u = L_u \cup \{a\} \), is diagnosable.

With this new lemma, Theorem 1 remains true. Moreover, if we redefine Definition 10 and Definition 11 as follows, also Theorem 2 remains true.

**Definition 17 (\( S_{\sigma\alpha} \) in \( \text{LTS}(\Pi_f) \)).** Let \( A(L_a) \) be a possibly diagnosable \( \text{LTS}(\Pi_f) \), then \( \forall 1 \leq i \leq m : \nexists \sigma, \alpha \in \text{traces}(A) : \sigma \in \text{traces}^{i}=_{\alpha}(A) \wedge f_i \notin \alpha : \sigma_{L_u} = \alpha_{L_u} \), we define:

- \( S_{\sigma\alpha} = \{ B \subseteq L_o | \sigma_{L_{\alpha} \cup B} \neq \alpha_{L_{\alpha} \cup B} \} \)
- \( S \) is a minimal cardinality set, such that

\[ \forall S_{\sigma\alpha} : \exists B \in S_{\sigma\alpha} : B \subseteq S. \]

**6.2 Predictability**

In some cases, e.g., in mission critical scenarios, it is important to achieve the prediction of a possible fault situation rather than a post-fault detection. For such scenarios we have to ensure that the fault is predictable, and we come to the problem of predictability. Predictability study is not new, it was first introduced in [2]. However, in [2] authors investigate only case of strongly predictable systems, ignoring the notion of safe predictability.

However, since predictability is about future, and future is non-deterministic, we have two types of predictability: safe predictability and strong predictability. Safe predictability refers to an observation of a sequence of events that may potentially end in a fault; while strong predictability refers to cases that will end in a fault (when the fault is unavoidable).

**Definition 18 (Safe predictability).** A \( \text{LTS}(L_f) \)

\[ A(L_a) = \langle Q, q^0, L, T, L_o, L_f \rangle \] is safely predictable if the following holds: \( \forall \sigma \in \text{traces}^\infty(A) \) if

\[ \exists a \in \text{traces}(A) : f \notin a \wedge \sigma_{L_u} \alpha \text{ then } \exists a' \in \text{traces}(A) : a \alpha a' \wedge f \in a'. \]

**Definition 19 (Strong predictability).** A \( \text{LTS}(L_f) \)

\[ A(L_a) = \langle Q, q^0, L, T, L_o, L_f \rangle \] is strongly predictable if the following holds: \( \forall \sigma \in \text{traces}^\infty(A) \) if

\[ \exists a \in \text{traces}(A) : f \notin a \wedge \alpha_L \text{ then } \forall a' \in \text{traces}^\infty(A) : f \alpha a' \text{ then } f \in a'. \]

In our example, from Figure 1-(A), let assume we have \( L_o = \{a, b\} \). Then, the system is safely predictable, since whenever we observe events \( a \) or \( b \) we know that we have the possibility to have a fault in the future. However, the system is not strongly predictable, since there is no sequence of observable events that unambiguously predicts either \( f_1 \) or \( f_2 \) occurrence. In particular, if we remove \( b \) from the list of observable events, the system is not safely predictable for all faults but it is safely predictable w.r.t. \( f_1 \). On the other hand, the example from Figure 1-(B) is clearly strongly predictable with a list of observable events like \( L_o = \{a\} \).

Within the defined framework, as in [2; 3] where similar results are presented, strong predictability implies diagnosability and safe predictability:

**Property 2.** Strong predictability implies diagnosability. It follows immediately from Property 1.

**Property 3.** Strong predictability implies safe predictability.

A predictability signature is defined as a set of observable events that, if occurred, always or potentially (depends on the type of predictability) bring the execution to a fault event.

**Definition 20 (Safe/strong predictable signatures).** Given a \( A(L_a) = \langle Q, q^0, L, T, L_o, L_f \rangle \in \text{LTS}(L_f) \), then:

- if \( A \) is safe predictable then its observable-safe-predictable-signatures (safe\_pr) is

\[ \text{safe\_pr} = \{ \sigma \in L_o^* | \exists a \in \text{traces}^\infty(A) : \sigma_{L_u} \subseteq \alpha_{L_u} \} \]

- if \( A \) is strongly predictable then its observable-strong-predictable-signatures (strong\_pr) is

\[ \text{strong\_pr} = \{ \sigma \in L_o^* | \forall a \in \text{traces}^\infty(A) : \sigma_{L_u} \subseteq \alpha_{L_u} : f \in \alpha \} \]

We can apply the previous algorithms directly for strongly predictable systems.

**Property 4.** Given a strongly predictable system \( A(L_a) \in \text{LTS}(L_f) \) with \( c \) as its correct behaviour (Definition 7) and strong\_pr as its strong-predictable-signatures (Definition 20), then:

(i) The algorithm presented in Theorem 1 reduces the set of observable events correctly, keeping the system strongly predictable.

(ii) The algorithm presented in Theorem 2 expands the set of observable events correctly, keeping the system strongly predictable.

We also can apply the algorithms for safe predictable systems, but it is necessary to adapt the notion of observable correct behaviour.

**Definition 21 (Secure correct behaviour).** Given a system \( A(L_a) \in \text{LTS}(L_f) \) we define the observable secure correct behaviour as

\[ \text{sc} = \{ \sigma \in L_o^* | \sigma \in \text{traces}^\infty(A) : \forall \alpha : \exists a \in \sigma \subseteq \alpha_{L_u} : \exists a' \in \sigma : f \in a' \} \]

This definition is an adaptation of that of observable correct behaviour (Definition 7 in Section 4.2). The idea is analogous to the previous one: the main novelty here is that sub-traces of correct traces can never be part of observable-safe-predictable-signatures. In this way we still keep the structural difference between correct behaviours and signatures.

**Property 5.** Given a safe predictable system \( A(L_a) = \langle Q, q^0, L, T, L_o, L_f \rangle \in \text{LTS}(L_f) \) with \( a \) as its secure correct behaviour (Definition 21) and safe\_pr as its safe-predictable-signatures (Definition 20), then:

(i) The algorithm presented in Theorem 1 reduces the set of observable events correctly, keeping the system safe predictable.

(ii) The algorithm presented in Theorem 2 expands the set of observable events correctly, keeping the system safe predictable.

We leave the properties in this section without proofs, since the proofs are analogous to Theorem 1 and Theorem 2.

**6.3 Extended fault model**

In this section we define an extended fault model, where a fault is formed by a specific fault sequence of events, that are not faults by themselves. Consider an example of driving a vehicle, where driving having doors open is a fault, while in
most other situations it is an absolutely legal and expected action. In this case, the fault is defined not by a faulty event but rather by a sequence of events that forms a fault. Furthermore, a fault sequence can contain any arbitrary events that do not contribute to the fault. For the vehicle example, we may have something occurred between opening the door and driving, and still, if the door is open, we are not allowed to drive. We define this problem as diagnosability problem in an extended fault model.

In the extended fault model the fault is defined as a sequence of events. The fault is considered to be occurring when the last event of the sequence occurs. Besides, events in the sequence are not required to occur one after another one, we may have other events happening in the meantime.

An extended fault is defined by a sequence of events, denoted \( \rho_f \). The set of fault executions is then defined as \( \{ \sigma \mid \rho_f \subseteq \sigma \} \), where \( \rho_f \subseteq \sigma \) means that the events from \( \rho_f \) happened in \( \sigma \) in order but not necessarily consecutively. So, if \( \rho_f = ab \) then the trace \( \sigma = cache \) is \( \in \sigma.r. \). We denote by \( \rho'_f \) the last event of the fault sequence \( \rho_f \), i.e., \( \rho'_f = last(\rho_f) \).

Definition 22 (Diagnosability with an extended fault model). An extended fault model system \( A \) is called diagnosable w.r.t. to a fault sequence \( \rho_f \) and a set of observable events \( L_o \) if exists \( n \in N : \forall \sigma \in traces^{\rho_f, \alpha}(A) \) if \( \alpha \in traces(A) : \sigma_{L_o} = \alpha_{L_o} \), then \( \rho_f \in \alpha \).

A signature of an extended fault, in a diagnosable system \( A \), is defined by a set of observable traces that contain the extended faults, i.e., \( \tau = \{ \sigma_{L_o} \mid \rho_f \subseteq traces^{\rho_f, \alpha}(A) \} \).

Theorem 1 and Theorem 2 remain true also for diagnosability with extended fault model, since definitions, theorems and proofs, obtained in Section 5, work at the level of signatures, without representing the nature of signatures explicitly.

In our example we can assume that an extended fault of the system is \( \rho_f = bb \), meaning that the system execution a fault if both 2 events are performed. It is easy to see that the system is diagnosable for the following set of observable events \( L_o = \{ a, d, c, e \} \). Applying Theorem 1, we may reduce the set to \( L_o = \{ a \} \). From the other side, from a set of observable events \( L_o = \{ d \} \) (which makes the system not diagnosable), we may expand it to a set \( L_o = \{ d, a \} \) using Theorem 2.

7 Conclusion

In the paper we discussed different levels of observability for diagnosable discrete-event systems. We mainly studied two approaches: first, we transform a diagnosable system into one with minimal observability and still diagnosable. Second, we transform a non-diagnosable system into diagnosable by increasing the observability of the system. We presented algorithms that implement our two approaches and we illustrated our propositions with an intuitive example through the paper. Moreover, we provided several extensions to the problem of reducing and expanding of observability in diagnosable systems. Furthermore, the provided framework deals with both classical faults and an extended fault model, as well as with other extensions in a uniform way.

In the future work we plan to further investigate various extensions to the diagnosability problem and see if our framework can be extended to deal with new types of problems. We also plan to evaluate the proposed algorithms against some real cases within the WS-Diamond project [14]. As a part of the implementation and evaluation process we want to have our framework to go distributed making it faster and more efficient. The important issue of fault isolation and control is ignored within the framework presented in the paper. In the future we plan to extend the proposed approach with controllable actions that allow us to isolate faults or, at least, perform some compensation activities to repair the system from the occurred faults.

References


[2] GEN, S., AND LAFORTUNE, S. Predictability in discrete-event systems under partial observation. In IFAC World Congress (Beijing, China, August 2006).


Compilation Techniques for Fault Detection and Isolation: A Comparison of Three Methods

Anibal Bregón*, Gautam Biswas**, Belarmino Pulido*

* Department of Computer Science. University of Valladolid. Valladolid, Spain
e-mail: {anibal,belar}@infor.uva.es

** Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville TN 37235, USA
e-mail: {gautam.biswas}@vanderbilt.edu

Abstract

The FDI and DX communities have developed complementary approaches that exploit structure and redundancy in the system model to solve the diagnosis problem. These approaches share a methodology that we call compilation techniques. This paper discusses compilation methods for nonlinear dynamic systems.

In this work, we compare and discuss three different compilation approaches, all developed from the bond graph modeling framework: the traditional analytical redundancy relations method developed by the FDI community, and possible conflicts and qualitative fault signatures that have been developed by the DX community. We describe the diagnosis algorithms for the three schemes, and then perform experimental studies to demonstrate their diagnostic ability for continuous, nonlinear systems.

Keywords: Continuous Fault Diagnosis, Bond Graphs, Temporal Causal Graphs, Possible Conflicts, Analytical Redundancy Relations

1 Introduction

Power generation, manufacturing, transportation, and domestic appliances are complex engineering systems that pervade every aspect of our daily lives. The needs for increased performance, safety, and reliability of these systems motivate the use of efficient fault detection, diagnosis and recovery mechanisms. The goal is to automatically detect and isolate faults and degradations, and recover to a normal operating mode, and if that is not possible, move the system to a safe operation mode to avoid damage and harm.

Several approaches have been developed to solve the problem of fault detection and isolation. Model-based diagnosis approaches are quite prevalent because they have the potential to overcome the device dependency problem, which greatly increases the cost of developing and deploying diagnosis systems. Model-based diagnosis techniques require accurate models of the system to ensure accurate analysis of system behavior in different operating regions.

Traditionally, two different communities have tackled the problem of model-based diagnosis: the Control Engineering community, have developed FDI approaches [Gertler, 1998; Patton et al., 2000], and the Artificial Intelligence community, have developed DX approaches [Hamscher et al., 1992; Reiter, 1987]. FDI and DX approaches to model-based diagnosis of dynamic systems employ different kinds of models, and different assumptions concerning robustness of the generated solution with regard to disturbances, modeling errors, and noise.

An important issue in the use of model-based diagnosis techniques (for both the DX and the FDI communities) is the ability to construct accurate models of the system. An approach that has been successful for modeling the dynamic behavior of physical systems are bond graphs [Karnopp et al., 2000]. Bond graphs provide an easy, intuitive way to build multi-domain energy-based models by combining system topology with a small set of component behavior processes. The topological structure of bond graph models also provides the infrastructure for developing effective and efficient fault diagnosis methods based on causal analysis that links component parameters to system variables using well-defined methods.

Different diagnosis approaches have been developed in the two communities; e.g., the methods of qualitative fault signatures [Mosterman and Biswas, 1999] and Possible Conflicts (PCs) [Pulido and Alonso-González, 2004] by the DX community, and Analytic Redundancy Relation (ARR) methods in the FDI community [Samantaray et al., 2006]. In this paper, we compare and discuss the three approaches. We start with a common modeling framework, bond graphs, describe the diagnosis algorithms for the three schemes, and then perform experimental studies to demonstrate their diagnostic ability for continuous, nonlinear systems.

All three algorithms employ compilation methods to derive transformed models from bond graphs that facilitate the fault isolation task. For example, the ARR and PC methods manipulate the constituent equations derived from the bond graph models to build analytical relations between process parameters, measurement variables, and input variables to the system. The qualitative fault signature method derives a causal structure called the Temporal Causal Graph (TCG) that captures the causal and temporal relations between process parameters and the measurement variables. Fault signatures are
derived using propagation methods in the TCG structure.

This paper is organized as follows. Section 2 briefly describes the bond graph modeling approach and the method for deriving causal information from the bond graph. The bond graph model of a nonlinear, controlled three tank system is derived to illustrate the modeling approach. Section 3 describes the theoretical background for each of the three methods, and the diagnosis schemes that are derived for each method. Section 4 then describes an experimental method for comparing the three approaches. Section 5 presents the conclusions and directions for future work in this paper.

2 Modeling Methodology

2.1 Modeling with Bond Graphs

Bond graphs are labeled, directed graphs, that present a topological methodology for modeling the dynamic behavior of physical systems. Bond graphs were developed by Prof. Henry Paynter (1959) as a powerful language for modeling systems across different domains. The main advantages of this approach can be summarized as:

- It is intuitive. It is based on the systems topology and includes the notion of causality.
- It is a generic modeling language and uses a common energy-based framework for modeling subsystems in different domains (mechanical, electrical, fluid, thermal, etc.).
- The standard mathematical models of dynamic system behavior, e.g., the state space and I/O formulations, can be easily derived from bond graph models.

The nodes of bond graphs represent components or submodels, while the edges, called bonds, represent ideal energy transfer paths between the component and submodel ports.

Bond graphs adopt the lumped parameter approach to modeling physical systems. Dynamic behavior can be defined as a function of energy exchange between components of a system (or between different systems) [Karnopp et al., 2000]. In bond graphs, the state of a physical system is defined by the distribution of energy among its components at any particular time. The dynamic behavior, and, therefore, the future behavior of the system, is defined by the current state and the energy exchange mechanisms. In the bond graph approach, differential equation that govern the energy exchange mechanisms are expressed in terms of power. Power is the product of two conjugated variables [Broenink, 1999]: effort (e), and flow (f). Examples of effort and flow variables are: force and velocity (in mechanical systems), voltage and current (in electrical networks), and pressure and volume flow (in hydraulic systems).

2.2 Building System Models

With bond graphs, we can build system models using a small set of primitive elements [Broenink, 1999], for example. (i) energy storage elements (C, I); (ii) dissipative elements (R); (iii) source elements (S_e, S_f), that add or remove energy from the system; and (iv) junctions (1, 0), that represent ideal energy connections for sets of elements.

The laboratory plant shown in figure 1 will be used for empirical studies for comparison of the three approaches. This plant resembles common features of a continuous industrial process. It is made up of three tanks \{T_1, T_2, T_3\}, a control loop acting on the input of T_1, and four valves V_0, V_1, V_2, and V_3. In the operation protocol selected for this plant, the opening of V_0 depends on the pressure on tank T_1, while valves V_1, V_2, and V_3 are always completely open.

![figure 1](image1)

Figure 1: Diagram of the laboratory plant.

In this plant we have four measurements: pressure on tanks T_1 and T_3 \{-\{P_1, P_2\}\}, in-flow into tank T_1 \{-\{F_1\}\}, and out-flow from tank T_3 \{-\{F_2\}\}. We consider seven different faults in the plant: leak in tanks T_1, T_2, T_3, and block in valves V_1, V_2, V_3, and in the input pipe.

In figure 2 the corresponding bond graph model of the plant is shown. In this work, bond graph models can be constructed visually using the Fault Adaptive Control Technology (FACT) modeling tool [Manders et al., 2006].

![figure 2](image2)

Figure 2: Bond graph model of the plant. Here, \(f(x)\) represents the function of the pressure in T_1 that controls the resistance \(R_{value}\) (control on the aperture of V_0).

2.3 Deriving Causal Structure

The causal analysis in the bond graph approach consists on determining the computational relations of the effort and flow variables associated with the bonds. This process can be done automatically [Antic et al., 1999], and it’s important not only for the equation generation, but also for testing the correctness of the bond graph model.

To automatically assign causality to a bond graph model, it is necessary to take into account the four different types of causal constraints [Broenink, 1999]: fixed causality (S_e,
$S_f$), constrained causality (0-junction, 1-junction), preferred causality ($C$, 1), and indifferent causality ($R$).

Temporal Causal Graphs, TCG, are an extended form of signal flow graphs for dynamic systems. They capture the causal and temporal relations between process parameters and the measurement variables in the system. TCGs can be directly derived from the bond graph model of the system [Mosterman and Biswas, 1999]. More formally, a TCG can be defined as [Roychoudhury et al., 2006]:

**Definition 1 (Temporal Causal Graph).** A TCG is a directed graph $< V, L, D >$, $V = E \cup F$, where $V$ is a set of vertices, $E$ is a set of effort variables and $F$ is a set of flow variables in the bond graph system model. $L$ is the label set $\{-1, 1, p, p^{-1}, pdt, p^{-1}dt\}$ ($p$ is a parameter name of the physical system model). The $dt$ specifier indicates a temporal edge relation, which implies that a vertex affects the derivative of its successor vertex across the temporal edge. $D \subseteq V \times L \times V$ is a set of edges [Narasimhan and Biswas, 2007].

To derive the temporal causal graph, a process of two steps is carried out [Mosterman and Biswas, 1999]:

- Generate a representation of the system as a directed graph encapsulating relations among power variables in the bond graph.
- Add temporal information and component parameters to individual causal edges to form the TCG.

In the resulting temporal causal graph, effort and flow variables will be represented as vertices, while relations between variables will be represented as directed edges.

Regarding components in the bond graph model, junctions and resistors, will define instantaneous magnitude relations. On the other hand, capacitors and inductors will define both magnitude and temporal effects on causal edges. In figure 3, the temporal causal graph of the three-tanks system (figure 1) is shown.

3 Model Based Diagnosis

We start with the definition of a fault as [Blanke et al., 2003]:

**Definition 2 (Fault).** A deviation of the system structure or the system parameters from the nominal situation.

and illustrate the generic model-based diagnosis approach in Figure 4. The diagnoser tracks system behavior using a model of the system. Comparison between this predicted behavior and the observed behavior in the system will generate a residual [Isermann and Ballé, 1997]:

**Definition 3 (Residual).** Residuals capture the difference between observed behavior of a system defined by measurements and the predicted behavior of the system derived from dynamic system models.

Ideally, the value of a residual in nominal situation is equal to zero, and non zero in faulty situations. The discrepancy detection module is designed to reliably detect non-zero residuals, and invoke the fault diagnosis module when this happens. The fault diagnosis module uses the structure of the residual to isolate faults using constraint analysis techniques.

GDE defines a generic diagnosis process in the DX community, and is focused on conflict detection (equivalent to residual activation [Cordier et al., 2004]). Classically, this process has been carried out by online dependency-recording schema (like ATMS [de Kleer, 1986]), but some problems have been reported. The most important is related to the labeling process of correctness assumptions, which needs to be done each time a new value is introduced.

Trying to avoid this problem, several research groups have looked for alternative methods. A set of these alternative
methods uses the structural description of the system to be diagnosed (this information is implicitly stated in the system description). Those methods perform off-line dependency-recording, and are known as compilation techniques. The main idea behind the approach is that analytic redundancy relations (ARRs), that capture relations between fault parameters and system residuals, can be derived off-line.

Nowadays, compilation techniques can be considered as a established alternative to online dependency-recording within the DX community (e.g., qualitative fault signatures [Mosterman and Biswas, 1999], or possible conflicts [Pulido and Alonso-González, 2004]). Similar concepts are used in the FDI community (Analytical Redundancy Relations [Samantaray et al., 2006]).

3.1 Model-based FDI using Analytical Redundancy Relations. The Diagnostic Bond Graphs approach

In the FDI approach, residuals can be obtained through analytical redundancy in the models. One way to obtain this set of residuals is computing the set of ARRs$^2$.

Definition 4 (Analytical Redundancy Relation). An Analytical Redundancy Relation [Cordier et al., 2004], is a constraint deduced from the model of the system containing only observed variables, and consequently, can be evaluated from any subset of the observed variables, OBS. This subset of observed variables will be directly related with a subset of faulty parameters in the system, and this defines the isolation properties of the ARRs.

Computation of Analytical Redundancy Relations is carried out by an elimination procedure. This procedure consists, in general, on finding the set of over-determined systems for diagnosis, which is obtained from the unique canonical decomposition of the structural description of the system 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 with the structural description of the system. Combination of just-determined systems together with redundant relations is the basis for an Analytical Redundancy Relation.

Similar concepts can be derived in the bond graph framework [Medjaher et al., 2005]. A bond graph model in which causality can (or can not) be completely assigned, corresponds with a just-determined (or under-determined) system. Regarding over-determined systems, they can be identified by identifying all observable sub-graphs in the bond graph model. Using these sub-graphs, ARRs can be computed.

The classical method to derive ARRs from bond graphs is based on a two steps process [Samantaray et al., 2006]:

- Generate residuals using the conservation laws at each 1 or 0 junction.
- Check (using previously generated residuals) to make sure that the new residual is structurally independent.

This classical method of ARR derivation from bond graphs faces two important problems [Samantaray et al., 2006]: first, computational cost for equation derivation and structural equivalence checking is not efficient for large systems, and second, the method can’t be applied when symbolic methods are not able to eliminate all the unknown variables (e.g. in presence of loops and non linear non-invertible constraints [Medjaher et al., 2005]).

To solve these problems, [Medjaher et al., 2005] and [Samantaray et al., 2006] propose a method based on carrying out a set of substitutions in the sensors. The basic idea is substitute current sensors by some sub-graphs. Those sub-graphs will be designed depending on the causality of the sensors, so that, whenever possible, sensor causalities will be inverted (in this way, decoupling of residuals will be guaranteed). Five different configurations for these sub-graph models are considered$^3$: Inverted causality in effort sensor, inverted causality in flow sensor, non-inverted causality in effort sensor, non-inverted causality in flow sensor, and, inversion of signal sensor, Ds, to signal source, Ss (this is special, and is used to deal with controllers in systems). The bond graph of the system with all these substitutions, and using preferred derivative causality, is called the Diagnostic Bond Graph (DBG).

In figure 5 an example of these concepts applied to the three-tank system is shown [Samantaray et al., 2006]. The resulting DBG has been obtained from the bond graph model of the system (figure 2). Here, the causality on the sensors has been inverted, so that the bond graph is now able to diagnose directly from the input variables of the system.

3.1 Model-based FDI using Analytical Redundancy Relations. The Diagnostic Bond Graphs approach

In the FDI approach, residuals can be obtained through analytical redundancy in the models. One way to obtain this set of residuals is computing the set of ARRs$^2$.

Definition 4 (Analytical Redundancy Relation). An Analytical Redundancy Relation [Cordier et al., 2004], is a constraint deduced from the model of the system containing only observed variables, and consequently, can be evaluated from any subset of the observed variables, OBS. This subset of observed variables will be directly related with a subset of faulty parameters in the system, and this defines the isolation properties of the ARRs.

Computation of Analytical Redundancy Relations is carried out by an elimination procedure. This procedure consists, in general, on finding the set of over-determined systems for diagnosis, which is obtained from the unique canonical decomposition of the structural description of the system 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 with the structural description of the system. Combination of just-determined systems together with redundant relations is the basis for an Analytical Redundancy Relation.

Similar concepts can be derived in the bond graph framework [Medjaher et al., 2005]. A bond graph model in which causality can (or can not) be completely assigned, corresponds with a just-determined (or under-determined) system. Regarding over-determined systems, they can be identified by identifying all observable sub-graphs in the bond graph model. Using these sub-graphs, ARRs can be computed.

The classical method to derive ARRs from bond graphs is based on a two steps process [Samantaray et al., 2006]:

- Generate residuals using the conservation laws at each 1 or 0 junction.
- Check (using previously generated residuals) to make sure that the new residual is structurally independent.

This classical method of ARR derivation from bond graphs faces two important problems [Samantaray et al., 2006]: first, computational cost for equation derivation and structural equivalence checking is not efficient for large systems, and second, the method can’t be applied when symbolic methods are not able to eliminate all the unknown variables (e.g. in presence of loops and non linear non-invertible constraints [Medjaher et al., 2005]).

To solve these problems, [Medjaher et al., 2005] and [Samantaray et al., 2006] propose a method based on carrying out a set of substitutions in the sensors. The basic idea is substitute current sensors by some sub-graphs. Those sub-graphs will be designed depending on the causality of the sensors, so that, whenever possible, sensor causalities will be inverted (in this way, decoupling of residuals will be guaranteed). Five different configurations for these sub-graph models are considered$^3$: Inverted causality in effort sensor, inverted causality in flow sensor, non-inverted causality in effort sensor, non-inverted causality in flow sensor, and, inversion of signal sensor, Ds, to signal source, Ss (this is special, and is used to deal with controllers in systems). The bond graph of the system with all these substitutions, and using preferred derivative causality, is called the Diagnostic Bond Graph (DBG).

In figure 5 an example of these concepts applied to the three-tank system is shown [Samantaray et al., 2006]. The resulting DBG has been obtained from the bond graph model of the system (figure 2). Here, the causality on the sensors has been inverted, so that the bond graph is now able to diagnose directly from the input variables of the system.

$^3$For a deeper description of each of these configurations we refer to the original papers.
### 3.2 Possible conflicts: a dependency-compilation technique for consistency based diagnosis

Consistency based approaches are commonly employed in model-based diagnosis methods employed by the Artificial Intelligence community. Possible conflicts, PCs for short [Pulido and Alonso, 2000; Pulido and Alonso-González, 2004], are those sub-systems that produce conflicts when faults occur within the Consistency Based Diagnosis framework [Reiter, 1987], i.e. minimal subsets of equations containing the analytical redundancy necessary to perform fault diagnosis [Pulido and Alonso, 2000].

The main idea behind the possible conflict concept is that the set of subsystems capable of producing a conflict can be generated off-line. The PC computation process is carried out in a three steps process:

- Generate an abstract representation of the system, as an hypergraph ($H_{SD}$). In this representation there is just qualitative information about constraints in the models, and their relationship to known and unknown variables in such models.
- Look for the Evaluation Chains:
  
  **Definition 5** (Evaluation Chain ($H_{ec}$)). An Evaluation Chain is a connected subsystem ($H_{ec} \subseteq H_{SD}$), with observed variables, that can be solved using local propagation criteria (to follow GDE computational framework), and that defines a over-constrained set of relations.

  To find the Evaluation Chains, all the partial sub-hypergraphs in $H_{SD}$ capable of generating an estimation of an observed variable or a double estimation over a non-observed variable, have to be found. As in GDE, we are interested in minimal conflicts; hence, only minimal evaluation chains, MEC, are considered to be useful.

  **Definition 6** (Minimal Evaluation Chain). A evaluation chain, $H_{ec}$, is minimal if there is no other evaluation chain $H'_{ec} \subseteq H_{ec}$.

  These Minimal Evaluation Chains are a necessary, but not sufficient, condition for a conflict to exist. Each Minimal Evaluation Chain, which is a partial sub-hypergraph of the original system description, needs to be solved using only local propagation criteria. In the third step, extra knowledge is added to fulfill that requirement. Each possible way a constraint can be solved, by means of local propagation, is specified. As a consequence, each minimal evaluation chain generates a directed and-or graph. In each and-or graph, a search for every possible way the system can be solved using local propagation, is conducted. Each possible way is called a Minimal Evaluation Model, or MEM, and it can be used to predict the behavior of a subsystem.

Since conflicts will arise only when models are evaluated with available observations, the set of constraints in a MEC with at least one MEM is called a Possible Conflict, PC.

**Definition 7** (Possible Conflict). The set of constraints in a MEC that give rise to at least one MEM.

As a consequence, each MEM describes an executable model, which can be used to perform fault detection. If a discrepancy between predictions from those models and current observations is found, the possible conflict would 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 [Reiter, 1987].

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 and Alonso-González, 2004], the set of MEMs generated with this approach is equivalent to the set of conflicts computed by the GDE.

We applied all these concepts to the example on figure 1. We used the bond graph model (figure 2) to generate the hypergraph of the plant (figure 6). As we can see, the hypergraph relates variables belonging to the same constraint. For example, if we look at the bond graph model, we can see that $e_1$ is related with $e_0$ and $e_2$ because of the properties of 1-junction, so these three variables are related in the hypergraph. In the same way, $e_1$ is related with $f_1$ because of the properties of the resistor element, and so on. Using this hypergraph, we obtained a set of four possible conflicts. In figure 7, we can see an example of a possible conflict found for the system. In the right part of the figure, the discrepancy node for this possible conflict, $e_6$, is represented. This node compares the value of the effort $e_6$ given by the sensor $P_1$ (that comes from the process), against the value of $e_6$ obtained through the MEM (represented in the left part of the figure). Propagation of values will be done from the lower to the upper part of the MEM. That is, with sensor $P_2$, we can compute effort $e_{16}$. Using efforts $e_{16}$ and $e_{12}$, we can compute effort $e_{14}$, and so on. Dashed lines in the MEM represent temporal constraints in the system.

In table 2, the resulting signature matrix is shown. Columns $PC_1$, $PC_2$, $PC_3$, and $PC_4$ show the theoretical activation values of the PCs in presence of the different kind of faults considered (shown in the first column). Column $I$ shows the isolation capabilities of the approach.

---

The reader to [Samantaray et al., 2006] or [Medjaher et al., 2005]
The fault signature matrix shown in Table 1 was derived following the method described in [Samantaray et al., 2006].

3.3 Temporal Causal Graphs. The Qualitative Fault Signatures Approach

This approach, uses the bond graph model of the system to systematically derive qualitative fault signatures for diagnosis that correspond to the substructures used in the ARR and PC methods. The intermediate structure derived is called the TCG, and an offline graph traversal procedure called the prediction step is applied to each potential fault parameter to generate its fault signatures [Mosterman and Biswas, 1999]:

- Prediction: for each component parameter, that is a hypothesized fault, the candidate prediction algorithm is invoked. This algorithm computes, in a qualitative way, residual values of the observed variables. The residual is expressed in terms of the magnitude (zeroth order derivative), slope (first order time-derivative), and higher order effects.

All deviation propagations start off as zeroth order effects (magnitude changes). When an integrating edge in the TCG is traversed, the magnitude change becomes a first order change, i.e., the first derivative of the affected quantity changes.

On applying this two steps process to all the possible faults that can arise in the system, we obtain the fault signature matrix for the system (Table 3). In this table, $P_1$, $P_2$, $P_3$, and $P_4$ columns represent the expected deviations (no change (0), or increasing or decreasing (+/-)) in the measurements, and in the slope or higher order effects, in the presence of faults. Column $I$ shows isolation capabilities of this approach.

4 Experimental Results

4.1 Results of the case study

The set of analytical redundancy relations of a system can be directly derived from a bond graph model. For the case study, the fault signature matrix shown in Table 1 was derived following the method described in [Samantaray et al., 2006].
In the table, the relation between the analytical redundancy relations, obtained using derivative causality, and the faults considered in the system, is shown. As we can see, the system is able to isolate the faults related with (see column $I$): capacitance in tanks $T_1$ and $T_3$, resistance in valve $V_3$, and resistance in pipe $R_{pipe}$.

Using the equations derived from the bond graph model, we computed the set of possible conflicts for the system using integral causality. The resulting fault signature matrix is shown in table 2. As we can see, the isolation capabilities for the possible conflict approach are equivalent to the isolation capabilities for the ARRs approach.

The fault signature matrix for the qualitative fault signature approach is shown in table 3. In this table we only show the magnitude change symbol and the first non-zero direction of change symbol. A * symbol indicates an indeterminate effect, i.e., there are at least two paths of the same order that propagate + and - effects, and the dominant effect is unknown.

The TCG approach shows better fault discriminability than the PC and ARR approaches.

5 Discussion and Conclusions

As we can see in the results, the topological structure of bond graphs provide the infrastructure for developing effective fault diagnosis methods based on the derivation of causal strokes. All the approaches studied, make use, in different ways, of the topological structure of bond graphs.

An important issue in the comparison (as we will see later) is the way each approach deals with temporal information. Whereas ARRs and PCs only model the causal, and the temporal information between variables and their derivatives [Blanke et al., 2003], the qualitative fault signatures approach uses a more informative structure, the temporal causal graph. The additional information helps relate the direction of change in a variable to a fault hypothesis.

Based in our empirical studies we can summarize the differences between the three algorithms as follows:

- As we can see in table 1, the ARR approach suffers from two important drawbacks. First, the system is unable to discriminate between a fault in the capacitance of tank $T_2$, resistance in valve $V_1$, and resistance in valve $V_2$. Second, the approach uses derivative causality in the system, and a previous process of inversion on the causality of the sensors. Derivative causality is used to avoid the estimation of initial conditions in the system, but using this kind of causality makes the process of estimation harder.

- Possible conflicts obtain equivalent isolation results as ARRs (see table 2) without carrying out the process of inversion on the causality of the sensors. Nevertheless, there are important differences between both approaches. PCs look for redundancy between observed and estimated variables (like the ARR approach), but also look for redundancy between two estimated variables that can be computed using observed variables. That is why the number of PCs obtained for the plant could be bigger than the number of ARRs. This additional consistency check provides the approach with an important advantage not shown in this work: the improvement in the isolation capabilities of the system if we take into account sensor faults.

Another important difference is that each possible conflict has at least one MEM. Different MEMs use different causality assignments. Solving a set of equations in different order can produce significant differences for non-linear systems.

- Regarding the qualitative fault signatures approach, in the table 3 we show the fault signature matrix obtained for the case study. Two important issues have to be considered here. First, this approach uses qualitative information to drive the diagnosis task. The use of this kind of information allows this approach to obtain a more informed fault isolation space than the PCs or the ARRs though all three models are derived from the same bond graph structure. Fault signatures take into account not only deviations in the measurements, but also the sign of deviations in the measurements, slopes, and higher order effects. As can be seen in the fault signature matrix on table 3, the approach not only isolates all of the faults as the ARRs and PCs, but it is also able to isolate a fault in the capacitance of tank $T_2$. Looking at this table, we see that the only difference between a fault in $C_{T_2}$ and a fault in $RV_1$ or $RV_2$, is in the different sign of the derivative (or higher order effects) in the sensor $P_1$.

A second an important issue within this approach is that the fault detection and diagnosis capabilities can be directly and automatically derived from the bond graph model. It is not necessary to carry out a previous process of inversion in the causality of the sensor.

Qualitative fault signatures approach is able to provide more temporal information than the other two approaches. ARRs and PCs use only instantaneous and differential constraints. Recently, several authors in FDI, DX and BRIDGE communities [Puig et al., 2005; Gelso et al., 2008] have proposed extensions to improve the fault detection and isolation stages based on ARRs or PCs, including temporal information about residuals. Comparing with these extensions, the qualitative fault signatures approach still has an important advantage: the ability to derive these structures automatically.

An important issue to be considered here is that the structure of each possible conflict can be seen as equivalent to a minimal subset of over-determined equations within the temporal causal graph. If we look at the possible conflict on figure 7 and the temporal causal graph on figure 3, we can clearly see this equivalence: to compute $e_{T_2}$, both approaches need the value of $C_{T_2}$ and $f_0$; then, to compute $f_0$, both approaches need $f_1$ and $f_{10}$ (in the TCG, the relation between $f_9$ and $f_{10}$ is labeled as $\pm$, consequently $f_9$ and $f_{10}$ are equivalent); and so on.

The main task ahead is consider these similarities and establish, within the BRIDGE framework, the paths to integrate techniques to improve the general performance of the diagnosis task. On one hand, possible conflicts encapsulate the causal relations of the system, but a great part of the temporal relations between variables is missed. Including this information, the isolation capabilities of the possible conflicts...
approach, could be improved. On the other hand, the qualitative fault signatures approach, uses the temporal and causal information included in the TCG, and obtains better results for the isolation task, but it cannot handle the identification task, which we propose using PCs in future work.

Another important issue is carrying out the comparative analysis with larger and more complex systems. Our goal is to extend the current studies to more complex systems and perform systematic complexity analysis of the algorithms to establish further similarities and differences between the three compilation approaches.

Acknowledgments: Anibal Bregon and Belarmino Pulido’s work have been partially supported by the Spanish Ministry of Education and Culture (MEC 2005-08498). Gautam Biswas’ work was partially supported by NASA grants NNX07AD12A and USRA 08020-013.

References
An Event-based Approach to Hybrid Systems Diagnosability

Matthew Daigle*† and Xenofon Koutsoukos† and Gautam Biswas†
*University of California, Santa Cruz, NASA Ames Research Center, Moffett Field, CA, USA
†Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
mdaigle@arc.nasa.gov

Abstract

Diagnosability is an important issue in the design of diagnostic systems, because it helps identify whether sufficient information is available to distinguish all the faults. Diagnosability of hybrid systems, however, is challenging, because mode transitions may occur during fault isolation. We present an event-based framework for hybrid systems diagnosis based on a qualitative abstraction of measurement deviations from nominal behavior. We derive event-based fault models that describe the possible measurement deviations sequences due to faults, which, coupled with the mode transition structure of the system, are used to automatically synthesize an event-based diagnoser for hybrid systems. We introduce notions of diagnosability for hybrid systems and show how the event-based diagnoser can be used to verify the diagnosability of the system. We apply our diagnosability analysis scheme to a real-world electrical power distribution system.

1 Introduction

Diagnosability relates to the ability of a diagnostic system to obtain unique diagnosis results given a set of observations. Therefore, it is an important property that affects many aspects of the design of diagnostic systems. Based on diagnosability, we can determine at design time if a set of sensors provide sufficient discriminatory evidence, and, if not, what additional sensors may be useful.

Many modern engineering systems are best modeled as hybrid systems, which combine continuous and discrete behaviors in a common framework. Yet, diagnosability analysis of hybrid systems has largely been ignored. The task is complicated, because the effects of faults may change from one mode to another. In discrete-event systems, diagnosability refers to obtaining a sequence of observable events that is unique enough to identify which failure has occurred [Sampath et al., 1995; Zad et al., 2003]. Diagnosability of continuous systems has also been well-studied [Travé-Massuyès et al., 2006], and can be seen in much the same way, if fault signatures are viewed as events [Cordier et al., 2006; Daigle et al., 2007a; Meseguer et al., 2008]. Diagnosability of hybrid systems is studied in [Benedetto et al., 2007], but is defined only as the ability to detect faults, and not to obtain unique isolation results. Hybrid systems diagnosability in the analytic redundancy relations framework is described in [Bayoudh et al., 2006], and accounts for the changes in fault signatures due to mode changes.

We adopt an event-based approach to hybrid systems diagnosability, where faults are viewed as unobservable events. Measurement deviations (i.e., fault signatures) and controlled mode changes form the set of observable events. As in [Sampath et al., 1995; Cordier et al., 2006], we say a system is diagnosable if the sequence of observable events after fault occurrence uniquely isolates the fault. Due to mode changes, diagnosability of hybrid systems is typically harder to achieve than for continuous systems. A hybrid system might be diagnosable within each individual mode, but mode transitions during the fault isolation process may lead to loss of diagnosability because fault effects could get masked. Therefore, we introduce the more practical notion of $Q$-diagnosability, in which diagnosability can be achieved by blocking or forcing certain controlled mode changes during fault isolation. We design event-based diagnosers, which are then used to verify the diagnosability properties of the system. We apply our diagnosability scheme to a subset of the Advanced Diagnostics and Prognostics Testbed (ADAPT) at NASA Ames, which is a complex electrical power distribution system.

The paper is organized as follows. Section 2 describes the qualitative fault isolation framework. Section 3 presents the event-based fault modeling approach. Section 4 formalizes diagnosability in our framework, and Section 5 describes the design of the event-based diagnoser and how it can be used to verify diagnosability. Section 6 presents the case study. Section 7 concludes the paper.

2 Qualitative Fault Isolation

We consider the problem of single fault diagnosis in hybrid systems. We represent faults as unobservable events, and consider both abrupt parametric faults, modeled as unexpected step changes in system parameter values, and discrete faults, modeled as unexpected changes in system mode. Nominal mode transitions can occur due to known external controller actions, or autonomous behaviors that depend on internal system variables. In this paper, we assume that autonomous mode changes do not occur during fault isolation. Autonomous modes changes and multiple faults...
can be incorporated in a more complex framework using the techniques presented in [Narasimhan and Biswas, 2007; Daigle, 2008]. This paper does not consider these extensions to focus on the notions of diagnosability for hybrid systems.

The hybrid diagnosis architecture is illustrated in Fig. 1. A hybrid observer, implemented as a switched extended Kalman filter, computes the expected behavior of the plant based on inputs $u(t)$ and controlled mode change commands $\sigma_q$ [Narasimhan and Biswas, 2007]. The difference between observed outputs, $y(t)$, and expected outputs, $\hat{y}(t)$, defines the residual, $r(t)$. The fault detector employs a statistical test of significance to robustly determine if the residual is nonzero using a sliding window technique [Biswas et al., 2003]. Measurement deviations from nominal behavior are abstracted via the symbol generator, and the event-based diagnoser uses the sequence of events formed by measurement deviations, $\sigma_m$, and controlled mode changes, $\sigma_q$, to isolate faults. In the following, we denote the set of modes as $Q = \{q_1, q_2, \ldots, q_P\}$, the set of faults as $F = \{f_1, f_2, \ldots, f_n\}$, and the set of measurements, which are time-varying signals obtained from the available sensors, as $M = \{m_1, m_2, \ldots, m_p\}$.

Measurement deviations are abstracted using qualitative $+$, $-$, and 0 values to form fault signatures [Mosterman and Biswas, 1999]. Fault signatures represent the immediate change in magnitude and the first nonzero derivative change. They also represent what is termed discrete change behavior, which describes whether the signal went from a nonzero to a zero value ($\exists$), a zero to a nonzero value ($\ni$), or had no zero/nonzero value changes ($\forall$) [Daigle et al., 2008].

**Definition 1 (Fault Signature).** A fault signature for a fault $f$ and measurement $m$ in mode $q$ is the qualitative magnitude, slope, and discrete change in $m$ caused by the occurrence of $f$, and is denoted by $\sigma_{f,m,q} \in \Sigma_{f,m,q}$. We denote the set of all fault signatures for fault $f$ and measurements $M$ in mode $q$ as $\Sigma_{f,M,q}$, where $\Sigma_{f,M,q} = \bigcup_{m \in M} \Sigma_{f,m,q}$.

If the fault signature for a fault $f$ and measurement $m$ can be uniquely determined, then $\Sigma_{f,m,q}$ is a singleton. In general, $\sigma_{f,m}$ may not be unique due to ambiguities in the qualitative arithmetic.

In addition to fault signatures, we also capture the temporal order of measurement deviations, termed relative measurement orderings [Daigle et al., 2007b], which refer to the intuition that fault effects will manifest in some parts of the system before others. Measurement orderings are based on analysis of the transfer functions from faults to measurements [Daigle et al., 2007b].

**Definition 2 (Relative Measurement Ordering).** If fault $f$ manifests in measurement $m_i$ before measurement $m_j$ in mode $q$, then we define a relative measurement ordering between $m_i$ and $m_j$ for fault $f$ in $q$, denoted by $m_i \prec_{f,q} m_j$. We denote the set of all measurement orderings for $f$ in $q$ as $\Omega_{f,M,q}$.

The fault signatures and measurement orderings can be automatically computed from a temporal causal graph representation that is derived from the system model, using a forward propagation algorithm to predict qualitative effects of faults on measurements and their possible sequences of deviations [Mosterman and Biswas, 1999; Daigle, 2008].

Given a sequence of observable events, i.e., measurement deviations and controlled mode changes, the fault isolation task consists of matching event sequences to hypothesized fault candidates. We define a candidate as a hypothesized fault and a hypothesized system mode.

**Definition 3 (Candidate).** A candidate $c$ is defined as $c = (f_i, q_i)$, where $f_i \in F$ is a hypothesized fault, and $q_i \in Q$ is a hypothesized current mode. The set of all candidates is denoted as $C$.

We wish to find candidates that are consistent with the sequence of observed events. A diagnosis is a collection of candidates that are consistent with the observations provided to the diagnoser after the time of fault occurrence, $t_f$.

**Definition 4 (Diagnosis).** At time $t \geq t_f$, a diagnosis $d \subseteq C$ is a set of candidates consistent with the observations made on the system during the interval $[t_f, t]$.

Fault isolation is performed incrementally, as new events are received. At each new event, the current diagnosis is reduced by eliminating candidates that are inconsistent with the new event, given the previous sequence of events. Ideally, the diagnosis will eventually reduce to a unique candidate.

### 3 Event-based Fault Modeling

In order to characterize diagnosability in our framework, we first need to define what it means for a candidate to be consistent with a sequence of observable events. We do this by modeling the possible sequences of measurement deviations that...
faults may cause in different modes as event traces. Candidate traces are then formed by a special composition of these individual traces to account for the interleavings of events caused by mode changes.

For a specific fault and mode, the combination of all fault signatures and relative measurement orderings yields all the possible ways a fault can manifest. We denote each of these possibilities as a fault trace.

**Definition 5 (Fault Trace).** A fault trace for a fault \( f \) over measurements \( M \) in mode \( q \), denoted by \( \lambda_{f,M,q} \), is a string of length \( \leq |M| \) that includes, for every \( m \in M \) that will deviate due to \( f \) in \( q \), a fault signature \( \sigma_{f,m,q} \), such that the sequence of fault signatures satisfies \( \Omega_{f,M,q} \).

Note that the definition implies that fault traces are of maximal length, i.e., a fault trace includes deviations for all measurements affected by the fault. We group the set of all fault traces into a fault language. The fault model, defined by a finite automaton, concisely represents the fault language.

**Definition 6 (Fault Language).** The fault language of a fault \( f \in F \) with measurement set \( M \) in mode \( q \), denoted by \( L_{f,M,q} \), is the set of all fault traces for \( f \) over measurements \( M \) in \( q \).

**Definition 7 (Fault Model).** The fault model for a fault \( f \in F \) with measurement set \( M \) in mode \( q \), is the finite automaton that accepts exactly the language \( L_{f,M,q} \), and is given by \( L_{f,M,q} \equiv (\Sigma, s_0, \Sigma, \delta, A) \) where \( \Sigma \) is a set of states, \( s_0 \in \Sigma \) is an initial state, \( \Sigma \) is a set of events, \( \delta : \Sigma \times \Sigma \to \Sigma \) is a transition function, and \( A \subseteq \Sigma \) is a set of accepting states.

The finite automata representation allows for the composition of the fault signatures and relative measurement orderings into fault models. The possible fault signatures and measurement orderings can be composed automatically to form the fault models based on the synchronization operation [Daigle et al., 2007a].

We need to define the candidate language in order to formally characterize consistency of candidates. Unlike fault traces, traces for candidates must contain both controlled mode change events and measurement deviation events. We denote the set of possible measurement deviation events as \( \Sigma_{\lambda} \), and the set of mode change events as \( \Sigma_{\mu} \).

When a controlled mode change occurs during fault isolation, the system model is updated, and a new nominal reference for symbol generation is computed. When a new measurement deviates in the new mode, current hypothesized candidates must match the predictions for these candidates in the new mode, ignoring previously deviated measurements, in order to still be consistent. There may be different possible modes of fault occurrence, depending on the history of control actions, therefore, the set of consistent candidates depends also on the expected mode of fault occurrence. Given this, we can now define a candidate trace. In the following, we denote the mode transition function of the system by \( \mu \).

**Definition 8 (Candidate Trace).** An event trace \( \lambda = \sigma \) is a candidate trace for \( c \equiv (f_i, q_i) \) and initial mode of fault occurrence \( q_0 \), if \( \sigma \subseteq \lambda' \in \lambda_{f,c,M,q} \), where \( q_i = \mu(f_i, q_0) \). An event trace \( \lambda = \lambda_{i} \sigma_{i+1} \) is a candidate trace for \( c \equiv (f_i, q_i) \) and initial mode of fault occurrence \( q_0 \), if \( \lambda_{i} \) is a candidate trace for \( (f_i, q_i) \), and if \( \sigma_{i+1} \in \Sigma_{\lambda} \) then \( \mu(\sigma_{i+1}, q_{i+1}) = q_{i+1} \), or if \( \sigma_{i+1} \in \Sigma_{\mu} \) then \( q_{i+1} = q_{i+1} \) and \( \sigma_{i+1} \subseteq \lambda' \in L_{f_i,M,M,q_{i+1}} \). A candidate trace for \( c \) with initial mode \( q_0 \) is denoted as \( \lambda_{c,q_0} \).

In other words, given a candidate trace, an extension of that trace by a measurement deviation event will also be a candidate trace for the same candidate, if the deviation is consistent with the candidate for the new mode (i.e., it is consistent with the fault language in the new mode). An extension of the trace by a mode change event, however, will be a candidate trace for a different candidate, namely, the one defined by changing the mode of the old candidate to the new mode.

Clearly, there may be an infinite number of candidate traces because controlled mode changes may keep occurring indefinitely. However, we are only concerned with maximal traces, i.e., those for which all measurements that will deviate in the current mode have deviated (as with fault traces).

**Definition 9 (Maximal Candidate Trace).** A candidate trace \( \lambda_{c,q_0} \) for \( c \equiv (f_i, q_i) \) is maximal if \( L_{c,M,q_{i+1}} = \emptyset \), where \( M_i \) is the set of deviated measurements for \( \lambda_{c,q_0} \).

Now, we can define the language of a candidate \( c \) with respect to an initial mode of fault occurrence \( q_0 \), \( L_{c,M,q_0} \), as the set of maximal candidate traces for \( c \) starting in \( q_0 \).

**Definition 10 (Candidate Language).** The candidate language for candidate \( c \), measurements \( M \), and initial mode of fault occurrence \( q_0 \), denoted as \( L_{c,M,q_0} \), is the set of all maximal candidate traces \( \lambda_{c,q_0} \).

The candidate language consists of all consistent maximal traces for the candidate. A maximal trace is consistent with a candidate if the mode of the candidate can be reached via the sequence of controlled mode changes in the trace, and the measurement deviations within the trace match the fault in the intermediate modes.

### 4 Diagnosability

Diagnosability is an important property of a system, because it enables us to make guarantees about the unique isolation of faults. We first provide definitions of distinguishability and diagnosability and then describe how these notions are captured in our event-based framework.

If two candidates will always produce different effects, we say they are distinguishable. For hybrid systems, we must define distinguishability with respect to an initial expected mode at the point of fault occurrence, as with candidate traces.

**Definition 11 (Distinguishability).** For an expected mode \( q \in Q \) at the point of fault occurrence, a candidate \( c_i \) is distinguishable from a candidate \( c_j \), denoted by \( c_i \neq_q c_j \), if for any possible sequence of controlled mode changes, \( c_i \) always eventually produces effects on the measurements that \( c_j \) cannot.

Candidate languages essentially capture the effects produced on the measurements for candidates, and thus characterize consistency of candidates with observed effects. Therefore, candidate languages can be used to establish distinguishability within our framework.
Lemma 1. For an expected mode \( q_0 \in Q \) at the point of fault occurrence, a candidate \( c_i \) is distinguishable from a candidate \( c_j \) given measurements \( M \) and possible modes \( Q \), if there does not exist a pair of candidate traces \( \lambda_{c_i,q_0} \in L_{c_i,M,q_0} \) and \( \lambda_{c_j,q_0} \in L_{c_j,M,q_0} \) such that \( \lambda_{c_i} \sqsubseteq \lambda_{c_j} \).

Proof. Assume \( c_i \) is not distinguishable from \( c_j \), i.e., \( c_i \sim q_0 c_j \) for initial mode of fault occurrence \( q_0 \). Then, by definition, starting in mode \( q_0 \), there must exist a maximal candidate trace by \( c_i \) that \( c_i \) can also produce. Therefore, there must exist some maximal candidate trace for \( c_i \), i.e., some \( \lambda_{c_i,q_0} \in L_{c_i,M,q_0} \), and some sequence of events for \( c_j \) that is not distinct from \( \lambda_{c_i,q_0} \). So, \( \lambda_{c_i,q_0} \) must be a candidate trace \( \lambda_{c_i,q_0} \) for \( c_i \). Therefore, if \( c_i \sim q_0 c_j \) then there exists some \( \lambda_{c_i,q_0} \in L_{c_i,M,q_0} \) and \( \lambda_{c_j,q_0} \in L_{c_j,M,q_0} \) such that \( \lambda_{c_i,q_0} \sqsubseteq \lambda_{c_j,q_0} \). By the contrapositive, if there does not exist \( \lambda_{c_i,q_0} \in L_{c_i,M,q_0} \) and \( \lambda_{c_j,q_0} \in L_{c_j,M,q_0} \) such that \( \lambda_{c_i,q_0} \sqsubseteq \lambda_{c_j,q_0} \), then \( c_i \sim q_0 c_j \).

Since candidate traces include mode change events, the candidate languages cover all possible sequences of controlled mode change events interleaved with measurement deviations. Therefore, checking distinguishability is equivalent to checking for common traces. So, if a maximal candidate trace, which is a sequence of controlled mode change events and measurement deviation events, for some candidate is a prefix for a second candidate, then if the first candidate occurs and produces that trace, the candidates cannot be distinguished, because no more measurements will deviate (since the trace is maximal).

In our framework, a system can be defined as follows.

Definition 12 (System). A system \( S \) is defined as \( (F,M,Q,L_{F,M,Q}) \), where \( F = \{ f_1, f_2, \ldots, f_n \} \) is a set of faults, \( M = \{ m_1, m_2, \ldots, m_p \} \) is a set of measurements, \( Q = \{ q_1, q_2, \ldots, q_q \} \) is a set of modes, and \( L_{F,M,Q} \) is the set of fault languages for each fault in each mode, i.e., \( L_{F,M,Q} = \{ L_{f,m,q} : f \in F, q \in Q \} \).

Using distinguishability, we obtain the following notion of diagnosability for a hybrid system.

Definition 13 (Diagnosability). A system \( S = (F,M,Q,L_{F,M,Q}) \) is diagnosable if for all \( c_i \) and \( c_j \) possible modes of fault occurrence \( q_0 \in Q \), where \( |c_i| \leq l \) and \( |c_j| \leq l \), \( c_i \sim q_0 c_j \).

If the system is diagnosable, then every two candidates are distinguishable using the measurements in \( M \). So, each sequence of measurement deviations and controlled mode changes we observe can be eventually linked to a diagnosis with a unique candidate. Hence, we can uniquely isolate all candidates of interest. If the system is not diagnosable, then ambiguities may remain after fault isolation, i.e., after all possible measurement deviations have been observed.

The definition of diagnosability allows making guarantees about fault isolation. Although controlled mode change events affect the diagnosis, since the diagnoser has no control over which controlled mode change events are issued, we cannot, in general, make any restrictions about when a mode change event will be issued. Thus, diagnosability in this sense is conservative. It may be possible, however, to avoid ambiguous diagnosis results if certain mode changes are blocked or executed. We define this as Q-diagnosability.

Definition 14 (Q-diagnosability). A system \( S = (F,M,Q,L_{F,M,Q}) \) is Q-diagnosable if for all \( c_i \) and \( c_j \) possible modes of fault occurrence \( q_0 \in Q \), where \( c_i \sim q_0 c_j \), then for every (maximal) \( \lambda_{c_i,q_0} \) where \( \lambda_{c_i,q_0} \sqsubseteq \lambda_{c_j,q_0} \), either there exists some sequence of controlled mode changes \( c_j \) where \( \lambda_{c_i,q_0} c_j \) is not maximal for any candidate, or for every \( \lambda_{c_i,q_0} c_j \) where \( \lambda_{c_i,q_0} \) is a sequence of controlled mode changes, \( \lambda_{c_j} \) is not maximal for any candidate.

If the system is Q-diagnosable, then for any trace that violates diagnosability, there is a sequence of controlled mode changes that can be applied such that the new trace is no longer maximal, i.e., more measurement deviations will occur, or for every partial trace that can become the violating trace via a sequence of controlled mode changes, the partial trace is not maximal. The first case corresponds to executing controlled mode changes to ensure more measurement deviations will occur. The second case corresponds to blocking a sequence of controlled mode changes such that we never encounter the violating trace in the first place.

5 Diagnoser Design

We construct from our fault models an event-based diagnoser, which is an extended form of a finite automaton. If our system is diagnosable, we can construct a diagnoser that uniquely isolates all candidates. If not, the constructed diagnoser will give ambiguous results for some maximal traces. But, if the system is Q-diagnosable, the ambiguous results can be avoided. We wish to use the diagnoser to help determine system diagnosability. The goal of the event-based diagnoser is, given a sequence of measurement deviation events and controlled mode change events, to determine which faults are consistent with the observed sequence. We define formally a diagnoser in our framework.

Definition 15 (Diagnoser). A diagnoser for a fault set \( F \), measurements \( M \), and modes \( Q \), is defined as \( D_{F,M,Q} = (S, I, \Sigma, \delta, A, D, Y) \) where \( S \) is a set of states, \( I \subseteq S \) is set of initial states, \( \Sigma \) is a set of events, \( \delta : S \times \Sigma \rightarrow S \) is a transition function, \( A \subseteq S \) is a set of accepting states, \( D \subseteq 2^S \) is a set of diagnoses, and \( Y : S \rightarrow D \) is a diagnosis map.

A diagnoser is a finite automaton extended by a set of diagnosticians and a diagnosis map. The initial states correspond to possible starting modes at the point of fault occurrence. A diagnoser takes events as inputs, which correspond to measurement deviations \( \sigma \in \Sigma \) and controlled mode changes \( \sigma \in \Sigma \). From the current state, a measurement deviation event causes a transition to a new state. The diagnosis for that new state represents the set of candidates that are consistent with the sequence of events seen up to the current point in time, i.e., it encodes the results that hypothesis generation and refinement would obtain.

The accepting states of the diagnoser correspond to a fault isolation result. We say that a diagnoser isolates a candidate if it accepts all possible valid traces for the candidate and the accepting states map to diagnoses containing the candidate.
Definition 16 (Isolation). A diagnoser $D_{F,M,Q}$ isolates a candidate $c$ if it accepts all $\lambda \in L_{c,M,q_0}$ for all nominal $q_0 \in Q$, and for each $s \in A$ that accepts a $\lambda \in L_{c,M,q_0}$, $c \in Y(s)$.

The notion of isolation gives us an indication of correctness of our diagnosers. If our diagnoser isolates all candidates, then it covers all possible observable fault traces, and, therefore, is constructed correctly. We also would like to achieve unique isolation of candidates, which is a stronger notion of isolation. For unique isolation, we require that the diagnoser isolates candidate $c$, but also that the corresponding accepting states uniquely determine $c$. This means that the diagnoser will accept all valid maximal candidate traces, but also that each trace will uniquely identify a single candidate.

Definition 17 (Unique Isolation). A diagnoser $D_{F,M,Q}$ uniquely isolates a candidate $c$ if it isolates $c$ and for each $s \in A$ that accepts some $\lambda_c \in L_{c,M,q_0}$, $c = Y(s)$.

Unique isolation relates to diagnosability, so it can provide us with guarantees about the ambiguity of the diagnosis results. If we can design a diagnoser that isolates all candidates of interest, then by examining the diagnoser we can determine if it uniquely isolates all candidates, and if so, that the system is diagnosable. If not diagnosable, we can also use the diagnoser to determine which traces result in ambiguities, and if possible, avoid those traces by permitting or prohibiting certain controlled mode changes during isolation, i.e., achieve $Q$-diagnosability.

Ultimately, we would like to systematically construct a diagnoser for a hybrid system $S$ that isolates all possible candidates. Further, we would like to show that if $S$ is diagnosable, then this diagnoser uniquely isolates all candidates. To do this, we use individual diagnosers for each fault-mode pair, and provide a composition operator to simultaneously compose all the individual diagnosers to a global diagnoser that isolates all the valid candidates.

First, we construct a diagnoser, $D_{F,M,q}^*$ for each single fault $f$ within each mode $q$ from $L_{f,M,q}$. Definition 18 ($D_{F,M,q}^*$). Given fault $f$ and mode $q$ for measurements $M$, with $L_{f,M,q} = (S, s_0, \Sigma, \delta, A, D)$, $D_{F,M,q}^*$ is defined as $(S, s_0, \Sigma, \delta, A', \{(f,q)\}, Y)$, where $Y(s) = \{(f,q)\}$ for all $s \in S$, and $A' = A$ if $s \neq s_0$, or $A' = \{s_0\}$ otherwise.

We simultaneously compose each of the individual diagnosers $D_{F,M,q}$. In incremental consistency checking, we project out measurements that have already deviated to obtain the set of consistent candidates for a new observation. For a diagnoser, the state-based form of the measurement projection operation on traces is formalized using boundaries and boundary transition functions.

Definition 19 (Boundary). The boundary for a state $s$ and deviated measurements $M_i$, $B_{M_i}(s)$, is defined as the set of all states $\delta(\lambda, s)$ such that $\lambda$ contains only measurement deviation events corresponding to those in $M_i$.

The boundary for a state $s$ is basically the set of states that may be transitioned to from $s$ via a trace $\lambda$ consisting of only events for measurements that have already deviated, i.e., measurements corresponding to the events for traces in the history of the state. Using the notion of a boundary, we define a boundary transition function with respect to a set of deviated measurements.

Definition 20 (Boundary Transition Function). The boundary transition function for an event $\sigma$, state $s$, and set of deviated measurements $M_i$, denoted as $\delta_{M_i}(\sigma, s)$, is a transition function that maps $\sigma$ and $s$ to some state $s'$, such that $s' = \emptyset$ if the cardinality of $\{\delta(\sigma, s_B) : s_B \in B_{M_i}(s)\}$ is not 1, or $s'$ is the single element in $\{\delta(\sigma, s_B) : s_B \in B_{M_i}(s)\}$, otherwise.

In other words, $\delta_{M_i}(\sigma, s)$ returns the unique state that can be reached from a boundary state of $s$ via $\sigma$, or $\emptyset$ if there are no states that can be reached or if the state is not unique. Because of the way the $D_{F,M,q}^*$ diagnosers are computed, the reachable state will always be unique or null, because traces with the same set of measurements map to the same state. In the following, we denote the measurements that have deviated in a state as $M(s)$.

We now describe a composition operator, $\Pi$, that simultaneously combines the $D_{F,M,q}^*$ for each possible $(f, q)$ pair. We split the mode set $Q$ into nominal modes $Q_N$ and faulty modes $Q_F$.

Definition 21 (II Composition). Given the set of all $(f, q)$ diagnosers, $D = \{D_{F,M,q}^* : f \in F, q \in Q\}$, $D_{F,M,q} \triangleq \Pi(D)$, where

- $I = \{s_0, s_0, \ldots, s_0, q_0(\emptyset, q) : q \in Q_N\}$
- $\Sigma = \Sigma \cup \Sigma_2 \cup \ldots \cup \Sigma_k \cup \Sigma_Q$
- $\delta(\sigma, s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i) = (s_{i+1,1}, s_{i+1,2}, \ldots, s_{i+1,k}, q_{i+1,1}, d_{i+1})$, where $\sigma \in \Sigma_Q$, all $s_{i+1,j} = s_{i+1,j}, q_{i+1} = \mu(\sigma, q_i)$, and $d_{i+1} = \{(f, \mu(\sigma, q)) : (f, q) \neq \emptyset \wedge (f, q) \in d_i\}$
- $\delta(\sigma, s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i) = (s_{i+1,1}, s_{i+1,2}, \ldots, s_{i+1,k}, q_{i+1,1}, d_{i+1})$, where $\sigma \in \Sigma_M$, $q_{i+1} = q_i, M_i = M(s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i)$, $s_{i+1,j} = s_{i,j}$ if $\delta_{M_i}(\sigma, s_{i,j}) = \emptyset$, or $\delta_{M_i}(\sigma, s_{i,j})$ otherwise, and $d_{i+1} = \{(f, q) \in d_i : \sigma \subseteq \lambda \in L_{f,M,q} \neq \emptyset \}$
- $S$ is the set of all $s$ reachable through $\delta$ from some $s_0 \in I$
- $A$ is the set of all $s_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i) \in S$ where there exists some $s_{i,j} \in s_i$, with some $s_{B,j} \in B_{M_i}(s_{i,j})$, such that $Y(s_{B,j}) \subseteq Y(s_{i,j})$
- $D$ is the set of all $d_i$ in each $(s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i) \in S$
- $Y((s_{i,1}, s_{i,2}, \ldots, s_{i,k}, q_i, d_i)) = d_i$

Theorem 1. The diagnoser $D_{F,M,Q}$ isolates all valid candidates.

Proof. Assume initial mode of fault occurrence $q_0$, candidate $c$, and trace $\lambda = \sigma_1 \sigma_2 \ldots \sigma_k \in L_{c,M,q_0}$. By the definition of a candidate trace, $\sigma_1$ is a candidate trace for $c' = (f, \mu(f, q_0))$ if $c_1 \subseteq X' \in L_{f,M,\mu(f,q_0)}$. Therefore,
(f, µ(f, q0)) ∈ h_{F,M}(σ_1), so by definition of ∧_l, the result-
diagnosis will contain (f, µ(f, q0)), so by definition of δ, the cor-
responding state is in S. Assume λ_1 is a can-
didate trace for c’ = (f, q_1) and has a corresponding state
s ∈ S. Then if σ_i+1 ∈ Σ_Q, λ_1σ_i+1 is a candidate trace for
(f, µ(σ_i+1, q_i)) and by definition of δ has a corresponding
state s ∈ S and the associated diagnosis has (f, µ(σ_i+1, q_i)).
If if σ_i+1 ∉ Σ_Q, then λ_1σ_i+1 is a candidate trace for (f, q_i) if
σ_i+1 ̸∈ h_{F,M}(µ(f, q_i)) and therefore by definition of a hy-
pothesis set, (f, q_i) ∈ h_{F,M}(σ_i+1), so by definition of ∧_l,
the diagnosis will contain (f, q_i) and by definition of δ, will
have a corresponding state in S. Therefore, there is a state for
any valid candidate trace. Given a state s ∈ S with a trace that
is maximal for c = (f, q_i), the substate of s that corresponds
to a state in D_{f,M,q_i} must have no measurement deviations
possible from its boundary, otherwise the trace would not be
maximal, and thus the boundary must contain an accepting state.

Further, we can show that if the system S is diagnosable,
then the diagnoser uniquely isolates all candidates.

**Theorem 2.** A system S = (F, M, Q, L_{F,M,Q}) is diagnos-
able if and only if D_{F,M,Q} uniquely isolates all valid can-
didates.

**Proof.** Assume S is diagnosable. Assume a c and λ ∈
L_{F,M,Q}. D^*_c S, M, Q isolates c, so must have corresponding ac-
cepting state s with c ∈ Y(s). Since S is diagnosable, there
cannot be a c’ where c and c’ are not distinguishable, by de-
finition of diagnosability. So, there cannot be some common
subtrace λ that maps to an accepting state that has both c’
and c. So, D^*_c S, M, Q uniquely isolates all c. Assume D^*_c S,
M, Q uniquely isolates all c. Then each possible fault trace λ
has an accepting state s where c ∈ Y(s). Thus, there cannot be
some c’, with trace X that reaches the same state, otherwise
c’ is in Y(s). Therefore, c and c’ are distinguishable, so S
is diagnosable. Thus S is diagnosable if and only if D^*_c S,
M, Q uniquely isolates all c.

6 Case Study

We apply the diagnosability framework to the Advanced
Diagnostics and Prognostics Testbed (ADAPT) deployed at
NASA Ames [Poll et al., 2007]. The testbed is functionally
representative of a spacecraft’s electrical power system, and
consists of three subsystems: (i) power generation, which in-
cludes two battery chargers, (ii) power storage, which con-
ists of three sets of lead-acid batteries, and (iii) power dis-
tribution, which consists of a number of relays and circuit
breakers, two inverters, and various DC and AC loads.

We consider a subset of ADAPT to demonstrate our ap-
proach, which includes a lead-acid battery, two relays, and
two DC loads. The battery is modeled by an electric cir-
cuit equivalent described in [Daigle, 2008] (see Fig. 2). The
battery supplies voltage to the relays through a parallel con-
nection, which in turn supply power to the two DC loads. The
selected measurements are the battery voltage, V_B(t), and
the currents through the relays, I_{L_1}(t) and I_{L_2}(t), i.e.,
M = {I_{L_1}, I_{L_2}, V_B}.

We consider faults in the battery, loads, relays, and sensors.
Common battery faults include loss of charge and resistance
increases brought about by battery use and age, which mani-
fest as a side effect of the chemical reactions. Loss of charge
is represented by a capacitance decrease, C_{01}, and an in-
ternal losses by R_{01}. Faults can occur in the system loads,
and these are represented by increases or decreases in their re-
stance values, R_{L1} and R_{L2}. For the sensors, we consider
bias faults, which produce abrupt changes in the measured
values manifesting as constant offsets. Sensor faults are la-
beled by the measured quantity they represent, e.g., V_{l1} rep-
resents a bias fault in the battery voltage sensor. We represent
discrete faults in Sw_1 and Sw_2 by fault events α and β, re-
spectively, where a subscript of 0 indicates a stuck-off fault,
and a subscript of 1 indicates a stuck-on fault.

6.1 Diagnosability Analysis

We denote the system mode as q_0ij and a controlled mode
change to q_0ij as σ_0ij, where i is the mode of Sw_1, and j is
the mode of Sw_2. We allow controlled mode changes that
switch the system from any one controlled mode to another,
i.e., Σ_Q = {σ_001, σ_002, ..., σ_071}. We restrict discrete faults
to only occurring from expected modes where a deviation
would be produced, e.g., σ_01 would not produce any devia-
tions if it occurred in a mode where Sw_1 was already on.

The fault signatures and relative measurement orderings
for the chosen faults are given in Table 1 for selected modes
(q_4+i indicates the signatures and orderings are valid for any
mode). The nonlinearities in the battery introduce ambigu-
ity in the qualitative signatures, and this is denoted by the * sym-
bol, e.g., a signature of 0* may manifest as 0+ or 0-.
Since the sensors are not part of any feedback loops in the
system, sensor faults affect only the measurement provided
by the sensor. The other measurements are not affected, and
so the corresponding fault signatures are denoted by 00, indi-
cating no change in the measurement from expected behavior.

Selected fault models for ADAPT are shown in Fig. 3.
Consider the fault model C_{R_{L2}+q_4}, shown in Fig. 3b. Note
that the M subscript is dropped in the notation. From the
orderings, the current through Load 1 must be the first to de-
viate, followed by the Load 2 current and battery voltage in
any order. The direction of the changes in I_{L2}(t) and V_B(t)
are unknown so both possibilities are represented. The in-
dividual diagnosers for the same faults are shown in Fig. 4.

Given any one mode, the system is diagnosable. After at

---

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.

![Figure 2: Electrical circuit equivalent for the selected subsystem.](image-url)
most two measurement deviations, a unique candidate can be isolated. However, over all modes, the system is not diagnosable. Fig. 5 gives a partial diagnoser for the system that illustrates this property, with $F = \{ C^-_0, R^+_L \}$ and initial mode $q_{11}$ with $\sigma_{q_{11}}$ and $\sigma_{q_{11}}$, being the only controlled mode change events. If $I^+_L$, $q_{11}$ occurs, we reach an accepting state that corresponds to a diagnosis with multiple candidates. After that event, both $C^-_0$ and $R^+_L$ are consistent. Since the state is accepting, it is possible that no new measurement deviations will occur to distinguish the faults. The resistance fault will have no visible effects on the rest of the measurements in this mode, because the source of the deviations is cut off, so we would have to wait infinitely long to verify $R^+_L$ as the true fault. Therefore, the system is not diagnosable. We can see, however, that the system is $Q$-diagnosable. If we prevent $\sigma_{q_{11}}$ from occurring, or change back to $q_{11}$ if it does occur, more measurements will deviate and we can distinguish the candidate uniquely. Additional diagnosability results that include multiple faults and autonomous mode changes, as well as diagnosis experiments, are reported in [Daigle, 2008].

### Table 1: Fault Signatures and Relative Measurement Orderings for the ADAPT Subsystem

<table>
<thead>
<tr>
<th>Fault</th>
<th>$V_B$</th>
<th>$I_L$</th>
<th>Measurement Orderings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V^+_B$</td>
<td>0, X</td>
<td>0, X</td>
<td>$V_B &lt; I_L$</td>
</tr>
<tr>
<td>$V^-_B$</td>
<td>0, X</td>
<td>0, X</td>
<td>$V_B &lt; I_L$</td>
</tr>
<tr>
<td>$R^+_L$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^+_L$</td>
</tr>
<tr>
<td>$R^-_L$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^-_L$</td>
</tr>
<tr>
<td>$(C_0, q_{11})$</td>
<td>0, X</td>
<td>0, X</td>
<td>$C_0$</td>
</tr>
<tr>
<td>$(R^+<em>L, q</em>{11})$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^+_L$</td>
</tr>
<tr>
<td>$(R^-<em>L, q</em>{11})$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^-_L$</td>
</tr>
<tr>
<td>$(R^+<em>L, q</em>{11})$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^+_L$</td>
</tr>
<tr>
<td>$(R^-<em>L, q</em>{11})$</td>
<td>0, X</td>
<td>0, X</td>
<td>$R^-_L$</td>
</tr>
</tbody>
</table>

### Figure 3: Selected fault models for ADAPT.

### Figure 4: Selected individual diagnosers for ADAPT.

7 Conclusions

We presented a systematic framework to create event-based diagnosers for hybrid systems. Using the diagnosers, diagnosability of the system can be analyzed. We introduced the notion of $Q$-diagnosability, in which unique isolation can be achieved if certain controlled mode changes are prevented or

Alban Graestien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Figure 5: Partial hybrid diagnoser for $F = \{C_0, R_{L1}^1\}$ and initial mode $q_{11}$.

executed during the fault isolation stage. We applied the technique to analyze the diagnosability of a subset of the ADAPT system.

Acknowledgments

This work was supported in part by grants NSF CNS-0615214, NASA USRA 08020-013, NASA NRA NNX07AD12A, and NSF CNS-0347440.

References


Using Model-Based Diagnosis for Bayesian Inference

Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA

Abstract

This paper shows how (discrete) Bayes Net inferences can be performed with a probabilistic Assumption-Based Truth Maintenance System (ATMS) as used in GDE-like model-based diagnosis algorithms. An advantage of this approach is that tasks that already require the capabilities of the ATMS (e.g., qualitative reasoning, diagnosis, natural language) can now also use the familiar Bayesian inferences. As the ATMS is designed to be incremental, many Bayesian inferences which would otherwise require full recomputation are accomplished directly. Some examples are: (1) new random variables, (2) new or changed CPTs, (3) changes in priors, (4) sensitivity analysis. In addition, the probabilistic ATMS supports exact and inexact inference for problems including MAP.

1 Introduction

Bayesian inference has become the dominant approach to probabilistic inference in AI. Truth Maintenance Systems are a common approach for handling ambiguous knowledge. Both approaches have been applied to very large problems. This paper shows how Bayesian inference can be performed within a probabilistic ATMS such that a system can exploit both approaches simultaneously. In addition, this paper sets the foundation for future work to explore how algorithms optimized for one approach might apply to the other.

Some of the advantages of encoding Bayesian inference using a probabilistic ATMS are:

- Can calculate Dempster-Shafer probabilities.
- Can determine upper or lower bounds on posterior probabilities when some priors are not provided.
- Complexity is independent of tree width.
- Every Bayes net can be encoded as an ATMS problem.

2 Background

There is a sizable literature on various approaches to integrating probabilistic inference into an ATMS. One goal of this paper is to bring these insights together into a consistent combined approach. GDE [de Kleer & Williams, 1987] used a probabilistic ATMS to support diagnostic inference which is described in the next section. This framework supported a long sequence of diagnostic research. [Provan, 1988; 1989] shows how a probabilistic ATMS computes Dempster-Shafer beliefs and provides an algorithm to compute probabilities of ATMS labels. [Goldman & Charniak, 1988] describes a probabilistic ATMS where the probabilities are primarily used for search control but does have a method to encode general CPTs. [D’Ambrosio, 1988; 1993] relates a probabilistic ATMS to support logic programming and provides another algorithm to compute probabilities of ATMS labels. [Poole, 1991; 1993b; 1993a] show how propositional and probabilistic inference can be naturally combined within one framework. The ATMS encoding for Bayes nets is directly related to the Multi-Linear Function encoding of [Darwiche, 2003].

3 ATMS


3.1 Logical Semantics of the ATMS

Every datum which the inference engine reasons about is considered a propositional symbol. The inference engine can refer to both the positive and negative instances of a symbol. These propositional literals are the ATMS nodes. Every important derivation made by the inference engine is stated as propositional formulas. Throughout this paper $\mathcal{F}$ refers to the set of formulas which have been communicated to the ATMS.

The inference engine designates a subset, $\mathcal{A}$, of the literals to be assumptions. $E \subset \mathcal{A}$ is an environment. Environment $E$ is inconsistent (called nogood) if $E \cup \mathcal{F}$ is not satisfiable. A
node \( n \) is said to be true (or hold) in environment \( E \) if \( n \) can be propositionally derived from \( E \cup \mathcal{F} \). A node \( n \) is said to be false in environment \( E \) if \( \neg n \) can be propositionally derived from \( E \cup \mathcal{F} \). Thus, given a node \( n \) and environment \( E \), \( n \) is either true, false, or unknown in \( E \). Unknown corresponds to the situation where adding neither \( n \) nor \( \neg n \) to \( \mathcal{F} \) would make \( E \) nogood. A nogood is minimal if it contains no others as a subset. The possible worlds \( \mathcal{W} \) is defined as:

\[
\mathcal{W} = \{ w \in 2^A | w \not\models \bot \}.
\]

The ATMS is incremental, receiving a constant stream of new nodes, assumptions, and formulas interspersed various queries concerning the environments in which nodes hold. To facilitate answering these queries an ideal ATMS attempts to compute probabilities without constructing the full distribution.

3.2 Implementation Note: Tries

Representing a node label as a set of sets yields very inefficient ATMS operations. Therefore, ATMS represent all labels as tries (see [Forbus & de Kleer, 1992] for a more detailed description how tries can be used in an ATMS). Some important properties are:

- Fixed literal ordering
- All negations are at the leaves of the trie
- Tries are canonical, so equivalence testing is direct
- Maintaining the ATMS consistency and Minimality properties is fast
- Test for subsumption (either testing whether an environment is subsumed by a trie, or removing all elements of the trie subsumed by an environment), the most common operations in incremental operations, are optimized.

This representation does not fit neatly in Darwiche and Marquis’ knowledge compilation map [Darwiche & Marquis, 2002] because it optimizes incremental operations so it has some of the properties of BDDs and some of prime implicates.

3.3 Probabilistic Semantics of the ATMS

We associate a lower bound \( p_L(n) \) and upper bound \( p_U(n) \) probability with each node. Logically, a node can either hold in a world, its negation hold, or neither. In probabilistic terms, worlds in which a node holds contribute to \( p_L(n) \), worlds in which a node’s negations hold contribute to \( p_U(n) \). If a node or its negation always hold, then \( p_L(n) = p_U(n) \). Let \( p_R(n) \) be the probability that neither the node nor its negation are known:

\[
p_U(n) - p_L(n) = p_R(n).
\]

By definition, for all ATMS assumptions, \( p_L(a) = p_U(a) \). In the Bayes Net encodings described later, \( p_L(n) = p_U(n) \) for all nodes.

Analogous to the purely logical case, all probabilistic inference follows from the prior probabilities of assumptions. Assumptions are presumed to be independent. Initially,

\[
p(w) = \prod_{a \in w} p(a) \prod_{a \in A \setminus w} p(a).
\]

\( p_L \) and \( p_U \) can be defined as follows:

\[
p_L(f) = \frac{1}{Z} \sum_{w \in W \text{ s.t. } f, w = f} p(w),
\]

\[
p_U(f) = 1 - \frac{1}{Z} \sum_{w \in W \text{ s.t. } \neg f, w = \neg f} p(w),
\]

\[
Z = \sum_{w \in W} p(w).
\]

3.4 Simple Example

![Figure 1: Labels after 4 formulas. Rectangles represent assumptions, ovals other nodes, and dark triangles represent formulas.](image)

Before examining ATMS algorithms, consider a simple example illustrated in Figure 1. \( A, B, C, D \) are assumptions and \( x, y, z \) are nodes. \( \mathcal{F} \) consists of 4 formulas \( A \land B \rightarrow x, B \land C \rightarrow y, x \rightarrow z, y \rightarrow z \). The ATMS labels are as shown. Suppose \( p(A) = 0.9, p(B) = 0.8, p(C) = 0.7, p(D) = 0.6 \). Figure 1 illustrates formulas in the TMS graphical style and Figure 2 illustrates the resulting joint probability distribution. Thus \( p_L(x) = 0.72, p_L(y) = 0.56, p_L(z) = 0.776 \). Blank cells in columns \( x, y \), and \( z \) that the truth of the variable cannot be proven or disproven for the corresponding world. As there are blank cells in each of the columns for \( x, y, z \) and no “F’s”, the upper bound of their probabilities is uniformly 1.

Suppose add the formula \( \bar{D} \land y \rightarrow \bot \) to \( \mathcal{F} \). The resulting ATMS structure is shown in Figure 3. The added formula renders two of the possible worlds inconsistent and the resulting probabilities have shifted to \( p_L(x) = 0.629, p_L(y) = 0.337, p_L(z) = 0.663 \).

4 Computing Probabilities from Labels Alone

The ATMS (and all Bayes Net algorithms) attempt to compute probabilities without constructing the full distribution.
The probabilistic ATMS algorithms compute the probability of a literal (or any formula) from its label. For example, the probability of \(x\) in Figure 1 can be computed directly from its label:

\[
p_L(x) = p(A = T \land B = T) = 0.9 \times 0.8 = 0.72,
\]

using the probability rule (assumptions are independent),

\[
p(X \land Y) = p(X) \times p(Y).
\]

Similarly,

\[
p_L(y) = p(A = T \land B = T) = 0.8 \times 0.7 = 0.56.
\]

Determining \(p(z)\) is more complicated because there are two environments in its label. The probability rule for sums is:

\[
p(X \lor Y) = p(X) + p(Y) - p(X \land Y).
\]

Thus,

\[
p_L(z) = p([A \land B] \lor [B \land C]),
\]

which expands to,

\[
p_L(z) = p(A)p(B) + p(B)p(C) - p(A)p(B)p(C) = 0.78.
\]

Consider what happens when a 5th formula is added. Adding the formula \(D \land y \rightarrow \bot\) results in the ATMS nogood:

\[

\neg B \lor \neg C \lor \neg D.
\]

This eliminates two rows of the joint probability distribution indicated by \(p^*(w) = 0\) in Figure 4. This nogood has two consequences on our probability calculation. First, we must make sure that the probability calculation does not include any inconsistent worlds. Second, the probabilities of nodes need to be normalized. Given a node \(a\) with ATMS label \(l\) and given known nogoods \(n\), \(p(a)\) given by:

\[
p_L(a) = \frac{p(l \land \neg n)}{p(\neg n)}.
\]

The probability of the nogood is easy to calculate:

\[
p_L(n) = p(B)p(C)p(D) = 0.336.
\]
For determining \( p_L(x) \), the numerator is calculated as follows:
\[
p([A \land B] \land \lnot B \lor \lnot C \lor \lnot D),
\]
expanding,
\[
p([A \land B \land \lnot C] \lor [A \land B \land \lnot D]),
\]
\[
p(A \land B \land \lnot C) + p(A \land B \land \lnot D) - p(A \land B \land \lnot C \land \lnot D) = 0.4176
\]
Thus,
\[
p_L(x) = \frac{0.4176}{1 - 0.336} = 0.629.
\]

5 Bayesian Inference

We now have enough ATMS machinery to render every Bayes Net as a probabilistic ATMS problem. This encoding exploits the ATMS oneof declaration. Although oneof constraints could be represented as a set of clauses, our ATMS implementation includes specific representations for oneof sets and no additional formulas need be added to the ATMS to represent them.

A (discrete) Bayes Net problem consists of:

- A set of random variables and a set of directed edges among variables.
- Each random variable has a finite set of values.
- The random variables and edges form a directed acyclic graph (DAG).
- Every variable \( X \) with incoming edges \( Y_1, \ldots, Y_n \) has an associated conditional probability table (CPT) for \( P(X|Y_1, \ldots, Y_n) \).

An encoding of a Bayes Net into the probabilistic ATMS is as follows. An oneof ATMS variable is created for every random variable. For each CPT, an ATMS assumption variable is created with values corresponding to each row of the CPT. In addition, an ATMS formula is added for each row of the table.

For every CPT row \( p(X = v|Y_1 = v_1, \ldots, Y_n = v_n) = y \) with assumption variable value \( a = k \) [de Kleer, 1986], a formula is added of the form:
\[
[Y_1 = v_1 \land \cdots \land Y_n = v_n \land a] \rightarrow A_{X=v}.
\]

In this encoding, \( p_L(X) = p_L(X) \) for every random variable \( X \) so we will simply write \( p(X) \) instead.

Consider the familiar Bayes Net of Figure 5. This Bayes Net has 3 CPTs and the figure shows a joint probability table for \( p(W, I) \) in Table 3. Figure 6 shows how this Bayes Net is represented in an ATMS. Figure 7 describes the derivation of the label of H=T (i.e., Holmes crashes).

\[
p(H = T) = p(A_{I=T}, H=T \land A_{I=T} \lor (A_{I=T}, H=T \land A_{I=T})).
\]

Which simplifies to:
\[
p(H = T) = p(A_{I=T}, H=T \land A_{I=T}) + p(A_{I=T}, H=T \land A_{I=T})
\]
\[
- p(A_{I=T}, H=T \land A_{I=T} \land A_{I=T})
\]

As the last term contains conflicting literals,
\[
p(H = T) = p(A_{I=T}, H=T)p(A_{I=T}) + p(A_{I=T}, H=T)p(A_{I=T}).
\]
\[p(H = T) = 0.1 \times 0.3 + 0.8 \times 0.7 = 0.59.\]

The Holmes graphical model is a polytree. It is well-known that exact inference in Bayes Nets which are polytree is linear. We will see later this is an instance of a more general ATMS property. For the moment, note that if the original Bayes Net is a polytree, the environments in any ATMS label are necessary disjoint and thus have the simplifying property that the probability of any label \( l \) can be computed by:
\[p(l) = \sum_{s \in L} p(s).\]

Notice also that the label of a node is a symbolic function for evaluating the probability of a node: Any of the priors can be changed without any label updates. However, this result comes at some cost. The label can grow exponentially in the depth of the polytree. This may not matter if labels are important to the application (e.g., we are finding all solutions). If the only purpose of the ATMS is to compute probabilities, then this disadvantage may outweigh other advantages. Later we discuss approximate inference which is used in many practical applications of the ATMS.

\[\begin{align*}
&0.7 \quad I=T \quad I=F \\
&I=F,H=F \quad I=T,H=F \quad I=T,H=T \quad I=F,H=T \\
&I=F,W=F \quad I=T,W=F \quad I=T,W=T \quad I=F,W=T \\
&I=T \quad I=F \\
&H=T \quad H=F \quad W=F \quad W=T \\
&0.2 \quad 0.8 \quad 0.1 \quad 0.9 \\
&0.1 \quad 0.2 \quad 0.8 \quad 0.1
\end{align*}\]

Figure 7: The computation for the ATMS label of H=T: \( \{ A_{I=F,H=T} \} \{ A_{I=T,H=T} \} \{ A_{I=T} \} \). The derivation is highlighted with heavier lines.

Let us return to the example of Figure 5. With no more evidence, both Holmes and Watson have crashed with probability 0.59. Consider the case where Watson has crashed. We learn \( W=T \). This will result in two contradictions (illustrated in Figure 8):
\[A_{I=F,W=F} \land A_{I=T} \rightarrow \perp, A_{I=T,W=F} \land A_{I=T} \rightarrow \perp.\]

There is never any reason to expand these to their clausal form. Given this information, the probability that Holmes has crashed rises to 0.765. This inference is performed in the probabilistic ATMS as follows. The label of H=T remains unchanged because none of its environments have become nogood. The general formula for the probability of a node with label \( l \) is:
\[p(l \land \neg n) = \frac{p(l)}{1 - p(n)}.\]

Which can be expressed without having to negate the set of all nogoods as:
\[p(l) - p(l \land n) \cdot \frac{1}{1 - p(n)}.\]

The denominator is just \( p(W = F) \) and is identical to \( H \)'s which we computed earlier to be 0.59. \( p(H = T) \) was calculated earlier:
\[p(H = T) = A_{I=T,H=T} \land A_{I=F} \lor A_{I=T,H=T} \land A_{I=F} = 0.59\]

The second term in the numerator is:
\[|A_{I=F,H=T} \land A_{I=F} \land A_{I=T,H=T} | = 0.59\]

which simplifies to:
\[A_{I=F,H=T} \land A_{I=F} \land A_{I=T,H=T} \land A_{I=T} \land A_{I=F} \land A_{I=T} = 0.59\]

As these are disjoint and substituting,
\[p(H = T) = 0.59 - (0.1 \times 0.3 \times 0.9 + 0.8 \times 0.7 \times 0.2) = 0.59\]

Note that \( 1 - p(n) \) in Bayes Net terms is the probability of evidence.

5.1 Computing Probabilities from Justifications Alone

Under Bayes encoding, the probability of a node can be computed from the justifications directly. This approach is particularly useful if the labels would otherwise become too large. Consider first the simpler case where there is no evidence (hence no nogoods) and the original Bayes Net is a polytree. Numerical values can be propagated from antecedents to consequents with two simple rules:

1. All inputs to a justification are multiplied.
2. All inputs to a node are summed.

Figure 9 illustrates the derivation of \( p(H = T) \) from the PATMS justification structure. The central advantage of this approach is that it is not necessary to construct a potentially exponential sized label.
5.2 Multi-Linear Functions

The ATMS encoding for Bayes Nets is directly related to the Multi-Linear Function encoding of [Darwiche, 2003]. Darwiche introduces a network parameter \( \theta_{i|\lambda} \) for each line of a CPT, and an evidence indicator \( \lambda_e \) for each value of a variable (the \( \lambda \)'s will either be 0 or 1). For example, the multi-linear function for Holmes crashing (before evidence) is:

\[
\begin{align*}
    f &= \lambda_{I=T} \lambda_{H=T} \theta_{I=T} \theta_{H=T} + \\
    &+ \lambda_{I=T} \lambda_{H=F} \theta_{I=T} \theta_{H=F} + \\
    &+ \lambda_{I=F} \lambda_{H=T} \theta_{I=T} \theta_{H=T} + \\
    &+ \lambda_{I=F} \lambda_{H=F} \theta_{I=T} \theta_{H=F}.
\end{align*}
\]

This is called the network polynomial. The ATMS encoding does not need to explicitly represent the \( \lambda \)'s because Darwiche’s multi-linear functions have the very strong constraint that the particular evidence indicators of a term are directly implied by the network parameters. For example, if \( \theta_{i|\lambda} \) occurs in the term it must also contain \( \lambda_e \). This observation reduces the worst-case label size (and multi-linear function size) by \( k^n \) where \( n \) is the number of variables and \( k \) is the number values each variable may have. (A more complicated encoding described later saves even more by observing, conversely, that the evidence indicators imply the network parameters, hence it is only necessary to represent ATMS assumptions representing the evidence indicators.)

The ATMS approach and Darwiche’s approach differ on how to condition the network. The ATMS approach uses the nogoods to represent the effects of evidence on a network, while Darwiche’s approach converts to Negated Normal Form.

5.3 Incremental Properties

The probabilistic ATMS is incremental. It also exploits lazy computation so that labels for nodes are constructed only when needed. It does not compute posterior probabilities for hidden variables nor for evidence variables that have not yet been observed. These are some of the incremental operations allowed on the encoded Bayes Net:

- Any prior probability can be changed at any time at no cost. The posterior probabilities are computed from the labels which do not change.
- Evidence variables (presuming they are not yet observed) can be changed to hidden variables or vice-versa at no cost.
- New random variables and their CPTs can be added at any time. Such additions will leave the existing label structure unchanged so that they are very efficient.
- Any random variable and their CPT can be removed at any time. This transaction is free because this just involves removing assumptions from the ATMS. However, if the random variable’s descendants supported some observed evidence, retracting a CPT may significantly change (better or worse) subsequent label and probability calculations for some nodes.
- Parents can be added or removed from a CPT for free. Again with the caveat that it may change future costs for computing posterior probabilities.

5.4 Exact MAP

We adopt the standard definition of MAP [Park & Darwiche, 2003]. Let the random variables be partitioned into three sets \( M \) (MAP variables), \( E \) (evidence variables) and \( S \). Given some evidence \( e \) (an instantiation of the evidence variables \( E \)), the MAP problem is to find an \( m \) (an instantiation of MAP variables \( M \)) which maximizes \( Pr(m, e) \):

\[
\arg \max_M \prod_{\phi \in \text{CPTs}} \phi_e.
\]

Where \( \phi_e \), the CPT (or potential if encoding a markov random field) restricted to evidence \( e \).

We can solve MAPs using model-based diagnosis as follows. We will make some simplifying assumptions for exposition. All random variables are binary valued (this is easily generalized), \( M \) are all root nodes of the network, and \( E \) contains all the remaining root nodes and only leaf nodes.

Model-based diagnosis algorithms are designed to maximize normality, thus each \( M \) is represented by \( \neg AB(x) \) (using the conventions of [Reiter, 1987; de Kleer, Mackworth, & Reiter, 1992]). The network DAG is further converted to a model-based diagnosis problem as follows.

- Each non-\( M \) root is an input signal.
- For each node in \( n \in M \), \( n \) is added to COMPS. The prior of \( \neg AB(n) \) will be the larger prior of its CPT. Another signal node \( m \) is created. The model for \( m \) is deterministic: its value is the same as \( n \)'s. For every outgoing edge from \( n \) to \( o \) in the Bayes net, a signal is added from \( m \) to \( o \). (If the random variable \( n \) has degree 1, we can optimize away \( m \).)
- For each node \( p \) directly connected to \( n \in M \), a component model is created. The most likely value represents \( \neg AB(n) \). The component model for correct behavior is a copy of \( p \)'s CPT restricted \( n \)'s most likely prior value. Conversely, the component model for incorrect behavior is a copy of \( p \)'s CPT restricted \( n \)'s most unlikely prior value.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Each leaf node is an output.

Each node connected to no node of M is modeled as a component, guaranteed to be correct, and whose model is the CPT.

Consider the simple case where non-root CPTs are deterministic (i.e., all the probabilities in those tables are 1 or 0). Solving MAP for such a system is equivalent to finding the most probable diagnosis. The resulting encoding corresponds to the case where all fault models are known (in the binary case, the individual component is always a wrong value). The methods embodied in GDE or Sherlock [de Kleer & Williams, 1989] can solve systems of very large size extremely quickly using A* search over M. The two key observations are that the nogood database contains all necessary information for search and that an admissible heuristic is simply computable from a partial instantiation of M.

Now consider the case where non-root CPTs are not deterministic. Again we use the same A* search framework avoiding nogoods. Unfortunately, evaluating an admissible heuristic is more complex. Consider the Bayes net illustrated in Figure 10. Without evidence the MAP is $m = \{B = F, C = F\}$.

![Figure 10: Bayes Net example to be encoded as a Model-Based Diagnosis problem.](image)

![Figure 11: Intuitive interpretation of the model-based diagnosis system corresponding to the Bayes Net of Figure 10.](image)

Figure 11 is an interpretation of the Bayes Net as a model-based diagnosis problem. We assume $A$ is always provided as evidence and $M = \{B, C\}$. Component $B$ can be viewed as a simple buffer, when $B = T$ it works correctly and when $B = F$ it works intermittently outputting a wrong value 0.9 of the time. Similarly, $C$ also is a buffer, but outputting a wrong value 0.8 of the time. Intuitively, the $AB(B) \land \neg AB(C)$ is the most likely diagnosis since it is more likely that $B$ outputs a wrong value than $C$.

Most model-based diagnosis algorithms incorporate evidence incrementally. In bayesian terms the network is conditioned after each piece of evidence. Without any evidence,

$$Pr(m) = \prod_{a \in m} Pr(a).$$

Given a single evidence assignment $x = c$, applying Bayes rule:

$$Pr(m, e \cup \{x = c\}) = \alpha Pr(x = c|m, e)Pr(m, e).$$

Where $\alpha$ is chosen so posterior probabilities sum to 1. The only unknown, $Pr(x = c|m, e)$ can be computed directly from the ATMS labels. As each $x = c$ is observed, it is added to the ATMS-encoded bayes net. Each addition will produce new nogoods and simplify existing labels. Therefore, $Pr(x = c|m, e)$ can be computed directly from the ATMS label corresponding to $x = c$.

Some model-based diagnosis algorithms are sound but incomplete which may yield misestimates. However, if the original bayes network is a polytree the mapped network is a polytree as well and almost all model-based diagnosis algorithms are complete for them.

5.5 Preferred Encoding

The previous encoding of Bayes Nets directly translates each CPT entry as an assumption with a distinct prior probability. A more complex mapping yields much better performance. Instead of creating an assumption for every combination of parent values, one need only create a single assumption for each possible value of the CPT variable itself. Then a formula is introduced for each combination of parent values. The shaded boxes in Figure 12 represent composite assumptions created instead of individual assumptions for each CPT entry. The upper left shaded box represents $A_{H=F} \equiv (A_{T=F, H=F} \lor A_{T=F, H=F})$. These composite assumptions do not have probabilities associated with them, however, wherever they occur in any label they will always occur with their parent assumptions which will always disambiguate the probability of the composition in the environment.

Although this encoding significantly reduces the number of assumptions and consequently the size of environments in labels, it does not reduce the number of environments that may appear in a label.

5.6 Approximate ATMS Inference

Just as exact inference in a Bayes Net can be too expensive, exact inference in an ATMS may result in prohibitively large labels. In order to limit label explosion the ATMS can be restricted to construct partial labels which are only complete with respect to certain foci [Forbus & de Kleer, 1988; de Kleer, 1992]. The challenge with this approach is to choose the relevant foci.

Fortunately, the model-based algorithm described earlier can be adapted to identify the necessary foci. In the prior algorithm, all labels and nogoods are completely computed prior for the MAP search. In [de Kleer, 1991] the MAP search alternates with the label update: (1) the best MAP solution is
found, given the current ATMS labels and skipping over all nogoods, (2) ATMS labels are updated only for that MAP solution, (3) \( Pr(m, e) \) is evaluated from the new label, (4) if the resulting probability is larger than the upper bound of any other partial instantiation of \( M \) the MAP solution is found otherwise repeat.

We saw earlier that to determine \( Pr(m, e) \) we need to evaluate the likelihood \( Pr(x = c|m, e) \). This likelihood can be evaluated as described in Section 4. It can be computed by simply marginalizing over the relevant label environments: \( Pr(x = c|m, e) = \sum_{s \in \text{label}(x = c)} a \leq m p(s) \).

6 Conclusions

The algorithms described in this paper have been completely implemented. We are currently comparing the performance of these algorithms on standard Bayes net benchmarks, but the results are not ready for publication.

This paper has presented an approach to for Bayes Net inference, exact or approximate, using an ATMS. The ATMS computes exact probabilities whether the Bayes Net is a poly-tree or not. It is incremental, allowing new CPTs to be added, removed or changed with limited cost. However, depending on the task, the ATMS may incur overall net effort increase. All Bayes nets can be directly encoded as a PATMS-based model-based diagnosis problem. However, few PATMS problems can be encoded directly as Bayes nets as it may contain loops and arbitrary dependencies between nodes. Hopefully, this paper will lead to more cross-fertilization of algorithms and ideas between the truth maintenance and Bayes Net approaches.

References


A framework for continuously estimating persistent and intermittent failure probabilities

Johan de Kleer, Bob Price, Lukas Kuhn, Minh Do, and Rong Zhou
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA

Abstract
Almost all previous work on model-based diagnosis has focused on persistent faults where the prior probability of component failure is provided by the manufacturer or estimated from fleet-wide service records. However, some of the most difficult to diagnose faults are intermittent. It is very difficult to isolate intermittent faults which occur with low frequency but yet at high enough frequency to be unacceptable. For example, a printer which prints one blank page out of a 1000 or a computer that spontaneously reboots once per day is unacceptable. Accurate assessment of intermittent failure probabilities is critical to diagnosing and repairing equipment. This paper presents an overall framework for estimating component failure probabilities which includes both persistent and intermittent faults. These estimates are constantly updated while the equipment is running. This paper also extends model-based diagnosis to systems where material instead of information in the form of voltages, currents, pressures is conveyed from one component to another.

1 Introduction
Most work on model-based diagnosis addresses isolating single persistent faults in physical systems where only information (voltages, currents, pressures, etc.) are conveyed among system components. This paper extends model-based diagnosis to include intermittent faults and to physical machines where material is being transferred from one system component to another. Thus we extend model-based diagnosis to the very difficult diagnostic task of troubleshooting manufacturing lines and plants.

In this paper we draw our examples from printers which should be considered as a manufacturing line which runs continuously and changes paper from one state (blank) to another state (marked on, stapled, bound, etc.). Extending our group’s work on developing self-aware printing machines [Fromherz, Bobrow, & de Kleer, 2003], we have designed and built the modular redundant printing engine illustrated in Figure 1. Such high-end reprographic machines operate more-or-less continuously providing a constant stream of observations and exception conditions. This paper addresses the challenge of estimating the probabilities of module failures from this data stream. These estimates are critical to avoiding modules which may fail (prognostics) as well as for sequential diagnosis.

Figure 1: Model of It consists of two towers each containing 2 internal printers (large rectangles). Sheets enter on the left and exit on the right. Dark black edges with small rollers represents a bidirectional paper path. There are three main paper (horizontal) highways within the modular printer. The printer incorporates 2 types of media handling modules represented by small lighter edge rectangles (described in more detail in Section 8). The motivation for this design is to continue printing even if some of the printers fail or some of the paper handling modules fail or jam.

Figure 2 illustrates the basic software architecture of our system. The basic task of the planner is to schedule sheets through the printer. The basic task of the diagnoser is to estimate module failure probabilities and provide diagnostic guidance to the planner. Both the planner and diagnoser operate with a common model of the machine state.

The reprographic machines receive a continuous stream of print jobs. Each print job consists of a sequence of sheets of paper. The planner constructs an optimal itinerary for each sheet of paper which specifies the full trajectory each sheet travels through the machine. These plans can consist of
dozens of modules. Failure is detected in two ways. First, a sheet arrives at a module while it is still handling a previous sheet. This will be detected by the module sensors and the module will immediately stop moving the paper (manifesting as a “jam”). Second, the system (Figure 1) has a scanner on the output so it can detect if the sheet has been damaged in any way.

Common kinds of failures are:

- A dog ear at one of the corners.
- Scuff marks on the paper caused by rollers (called nips) gripping the paper too tightly.
- The leading edge of the paper as it moves through the system may encounter a protrusion. (Leading edge damage.
- Paper is crumpled or shredded inside the machine.

These systems have some striking differences from the commonly explored digital circuits analyzed in most of the model-based diagnosis literature:

- Most errors cannot be masked or cancelled out. A damaged sheet cannot be repaired by the machine.
- The sheet may be touched by the same module more than once.

We note an itinerary and its outcome by the sequence of modules touched by the paper followed by Fail or Success. For example the itinerary in which a sheet passes through modules A,B,C,D,E,B,C and moved to the output tray without damage is represented as (A,B,C,D,E,B,C,Success). The itinerary in which a sheet passes through modules A and B and then jams in C is represented as (A,B,C,Fail).

### 2 Outline and Assumptions

In this paper we provide solutions for all combinations of multiple and persistent faults. Figure 3 illustrates the possibilities.

Each itinerary consists of a sequence of modules $m_1, \ldots, m_k$. We adopt the counting convention from [Abreu, Zoeteweij, & van Gemund, 2006] and associate four counters with each module $m$:

- $m_{11}$: number of plans where $m$ was used and failed.
- $m_{10}$: number of plans where $m$ was used and succeeded.
- $m_{01}$: number of plans where $m$ was not used and failed.
- $m_{00}$: number of plans where $m$ was not used and succeeded.

#### Figure 3: Fault combinations considered in this paper

<table>
<thead>
<tr>
<th>Used</th>
<th>Fail</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{11}$</td>
<td>$m_{01}$</td>
<td>$m_{10}$</td>
</tr>
</tbody>
</table>

The following simplifying assumptions apply for our reprographic engines:

- Every faulty module output is observable. (Catastrophic fault assumption.) Any damage to a piece of paper cannot be rectified by later modules. This assumption does not hold for digital systems where internal faulty signals can be masked to produce correct output. Our approach still applies for such systems but requires more reasoning to determine whether a faulty output is masked. (See [de Kleer, 2007].)
- Fault probabilities are stationary. Our approach can be easily extended to accommodate slowly drifting probabilities through discounting.
- Bad module produces bad outputs. This holds in the reprographic domain but not in digital domains in which case more reasoning is required to estimate posterior probabilities.
- Paper cannot damage a module. Most applications of model-based diagnosis presume signals cannot damage the system. However this does not hold for production lines which transport heavy objects as a misrouted object could damage the machine itself. Fortunately, in reprographic machines the relatively fragile paper is always what gets damaged.
- Observations do not affect machine behavior. This assumption is made in most approaches to diagnosis.
- All faults are distinguishable. This is simply for exposition: as in digital circuits, indistinguishable faults are collapsed.

These assumptions hold in a broad range of systems. The only input our approach requires is the sequence of itinerary-outcome pairs where the itinerary is expressed as a set of modules. For example, printers, manufacturing plants, bottling plants, and packaging plants can exploit our approach.
3 Single Persistent Fault

This case follows from GDE [de Kleer & Williams, 1987]. Let \( p(M) \) be the probability the module is faulted. The sequential bayesian filter [Berger, 1995] is:

\[
p(M|O, U) = \alpha p(O|M, U)p_{-1}(M).
\]

Where \( \alpha \) is chosen so that the posterior probabilities sum to 1 (presuming we start with the knowledge there is a fault). Let \( U \) be whether the module was used in the plan that produced the observation. \( p(O|M, U) \) is 1 in situations where \( m_{01} \) or \( m_{11} \) are incremented, otherwise it is 0. Namely, if the module is not used in a failing itinerary it is exonerated by the single fault assumption, and if the module is used in a successful plan it is exonerated because we assume that every faulty output is observed. Figure 4 illustrates the possibilities.

\[
p(O|M = m, U)
\]

Figure 4: Summary of the observation function in the single fault persistent case. Note that when diagnoses can have multiple faults, the test for whether a diagnosis is used generalizes to whether any of its models are used in the current itinerary.

Assume that at \( t = 0 \) all modules fail with prior probability \( p_0 = 10^{-10} \). Consider the arrangement of modules in Figure 5. Assume the sequence of itineraries: (A,B,C,D,E,F,Fail), (A,B,C,Success), (E,F,Success). After the (A,B,C,D,E,F,Fail) itinerary, one of the 6 modules must be faulted. As the priors are all equal, each module must be faulted with probability \( \frac{1}{6} \). As we assume faults are persistent and all faults are manifest, a successful itinerary exonerates all the modules of the itinerary. Thus the itinerary (A,B,C,Success) indicates that A,B and C are all working correctly. Finally, the itinerary (E,F,Success) exonerates modules E and F. Therefore, D is faulted with probability 1 (see Table 1).

<table>
<thead>
<tr>
<th>O</th>
<th>Fail</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Not Used</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: The resulting posterior probabilities \( p(M = m|O, U) \) over one sequence of itineraries. One persistent fault.

\[
\begin{array}{cccccccc}
\text{m} & m = \text{A} & m = \text{B} & m = \text{C} & m = \text{D} & m = \text{E} & m = \text{F} \\
\hline
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

4 Single Intermittent Fault

This case extends the model for intermittent faults presented in [de Kleer, 2007] which was informed by [Koren & Kohavi, 1977]. In the case of intermittent faults, \( p(O|M, U) \) is still 0 in case \( m_{01} \) and 1 in case \( m_{11} \). Otherwise, we need to estimate \( p(O|M, U) \) using the counts. The probability that module \( m \) produces an incorrect output if faulted is calculated as follows:

\[
\frac{m_{11}}{m_{11} + m_{10}}.
\]

(The denominator can never be 0 as will be described later.) Let \( p_0(M) \) be the prior probability that \( m \) is faulted. Given a particular observation \( O \), Bayes rule gives:

\[
p_1(M|O, U) = \alpha p(O|M, U)p_0(M).
\]

\( U \) represents whether the module was used in the plan. The observation function \( P(O|M, U) \), is estimated from the counts \( m_{ij} \). If the observation is a Failure and \( m \) is used, then:

\[
p(\text{Fail}|M = m, U) = \frac{m_{11}}{m_{11} + m_{10}},
\]

and if is Success and \( m \) is used, then:

\[
p(\text{Success}|M = m, U) = \frac{m_{10}}{m_{11} + m_{10}},
\]

otherwise as \( m \) cannot affect \( o \), if \( m \) good,

\[
p(\text{Success}|M = m, U) = 1,
\]

otherwise,

\[
p(\text{Fail}|M = m, U) = 0,
\]

(captures the single fault assumption). Figure 6 summarizes the 4 possibilities.

After many iterations of Bayes rule, intuitively,

\[
p_t(M|O) = \alpha^g p(\text{good})^g p(\text{bad})^b p_0(M),
\]

where there are \( g \) observations of \( m \)-used good behavior and \( b \) observations of \( m \)-used bad behavior. Formally:

\[
p_t(M|O, U) = \begin{cases} 0 & \text{if } m_{01} > 0 \\ \alpha w p_0(M) & \text{otherwise} \end{cases}
\]

where,

\[
w = \left[ \frac{m_{10}}{m_{11} + m_{10}} \right]^{m_{10}} \left[ \frac{m_{11}}{m_{11} + m_{10}} \right]^{m_{11}}.
\]
Consider again the arrangement of modules in Figure 5 and 3 itineraries: (A,B,C,D,E,F,Fail), (A,B,C,Success), (E,F,Success). The probabilities are updated as follows: After the first observation all \(m_{11}\) counters are 1 and the rest 0, therefore \(w's\) are 1. After observing (A,B,C,Success) the counters \((m_{00}, m_{01}, m_{10}, m_{11})\) for \{A, B, C\} are all 0, 0, 1 and the counters for \{D, E, F\} are all 1, 0, 0, 1. Therefore, \(w = \frac{1}{3}\) for \{A, B, C\} and 1 for the rest. We observe (E,F,Success) next. The counters for \{A, B, C\} are all 1, 0, 1, 1. The counters for \{D, E, F\} are now: 1, 0, 0, 1 and the counters for \{E, F\} are: 1, 0, 1, 1. Now suppose itinerary (A,B,C,D,E,F,Success) repeats for 7 iterations. Table 2 illustrates how the posterior probabilities evolve.

Table 2: The resulting posterior probabilities \(p(M = m | O, U)\) over one sequence of itineraries. One intermittent fault.

<table>
<thead>
<tr>
<th>(t)</th>
<th>(m = A)</th>
<th>(m = B)</th>
<th>(m = C)</th>
<th>(m = D)</th>
<th>(m = E)</th>
<th>(m = F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
</tr>
<tr>
<td>2</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>3</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
</tr>
<tr>
<td>4</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
<tr>
<td>10</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Working with the same 6 modules, consider a slightly more realistic example. Assume that the prior probabilities of intermittent failures are equal for all the modules. Consider the case in which module D is intermittently faulted and damages one out of every 1001 sheets (starting with sheet 1001). Suppose that the printer repeatedly executes the itineraries: (A,B,E,F), (C,B,E,D) (A,B,C) (E,F,D). After seeing 2000 itineraries the counts for A,F,C and D are \(m_{10} = 1000, m_{11} = 0\) and counts for B and E are \(m_{10} = 1500, m_{11} = 0\). Suppose D damages the sheet during the itinerary (C,B,E,D). By the single fault assumption, modules A and F are exonerated and their posterior probability of failure is now 0. The \(w\) for modules B and E are now:

\[
\frac{1500}{1501} \cdot \frac{1}{1501} = .000245. \tag{4}
\]

The term is higher for C and D as we have observed fewer samples of good behavior:

\[
\frac{1000}{1001} \cdot \frac{1}{1001} = .000368. \tag{5}
\]

Normalizing, the posterior probability for B, and E failing are: 0.2 and for C and D: 0.3. Suppose we see no errors in the next 2000 itineraries. Then, D causes a sheet in itinerary (D,E,F). By the single fault assumption, modules B and C are now exonerated. The values for \(w\) for D and E are now:

\[
\left[ \frac{2000}{2002} \cdot \frac{2}{2002} \right]^2 = 1.352 \times 10^{-7}, \tag{6}
\]

\[
\left[ \frac{3000}{3002} \cdot \frac{2}{3002} \right]^2 = 6.01 \times 10^{-7}. \tag{7}
\]

Normalizing \(p(D|O) = 0.7, p(E|O) = 0.3\).

Table 3: The resulting posterior probabilities \(p(M = m | O, U)\) over a more complex sequence of itineraries. One intermittent fault.

<table>
<thead>
<tr>
<th>(t)</th>
<th>(m = A)</th>
<th>(m = B)</th>
<th>(m = C)</th>
<th>(m = D)</th>
<th>(m = E)</th>
<th>(m = F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2003</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In practice faults never occur with such regularity as in Table 3. Instead, every sequence of itineraries will yield different posterior probabilities.

As can be seen in this example, the restriction to single faults is a very powerful force for exoneration. All the modules not exonerated will have the same \(m_{11}\) count. This results from the fact that under the single fault assumption, only modules that been used in every failing run remain suspect. Hence they have the same \(m_{11}\). In our example, \(m_{11} = 1\) in equations 4 and 5. After more observations, \(m_{11} = 2\) in equations 6 and 7.

4.1 Incorporating prior counts

So far we presume nothing is known about the counts prior to making observations. If counts are initially 0, then the denominator of equation 3 will be 0. One possible approach to avoid this is Laplace’s adjustment: make all initial counts 1, which is equivalent to assuming a uniform prior over \(p(m)\). Another approach which we utilize in this paper is to observe \(m_{00}\), and normalize it with the \(m_{10}\) count. This results in the following.

\[
\frac{1500}{1501} \cdot \frac{1}{1501} = .000245. \tag{4}
\]

In practice faults never occur with such regularity as in Table 3. Instead, every sequence of itineraries will yield different posterior probabilities.

As can be seen in this example, the restriction to single faults is a very powerful force for exoneration. All the modules not exonerated will have the same \(m_{11}\) count. This results from the fact that under the single fault assumption, only modules that been used in every failing run remain suspect. Hence they have the same \(m_{11}\). In our example, \(m_{11} = 1\) in equations 4 and 5. After more observations, \(m_{11} = 2\) in equations 6 and 7.

One important detail we leave out of the examples for conciseness is that if a module has operated perfectly for very

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
large counts it takes too many failing samples before its posterior probability rises sufficiently to be treated as a leading candidate diagnosis. Therefore, for our application, we apply a small exponential weighting factor \( \lambda \) at every increment such that counts 100,000 in the past will have only half the weight of new samples (\( \lambda = 0.99999 \)).

5 Multiple Persistent Faults

Instead of modules, consider diagnoses \( d \) which assign ‘good’ or ‘faulted’ to every system module. The number of possible diagnoses will be exponential in the number of modules. In practice, we only consider the more probable diagnoses, but for the moment consider the general case.

Analogous to the single persistent fault case:

\[
p_t(D|O, U) = \alpha p(O|D, U)p_{t-1}(D).
\]

To determine the prior probability of a diagnosis \( p_0(D) \) we presume modules fail independently:

\[
p_0(D) = \prod_{g \in \text{good}(D)} p_0(g) \prod_{b \in \text{bad}(D)} (1 - p_0(b)).
\]

It remains to determine \( p(O|D, U) \). If all the modules used in an itinerary are a subset of the good modules of a diagnosis \( d \), then \( p(\text{Fail}|D = d, U) = 0 \) and \( p(\text{Success}|D = d, U) = 1 \). In every remaining case (i.e., if any of the used modules are bad in \( d \)), then \( p(\text{Success}|D = d, U) = 0 \) and \( p(\text{Fail}|D = d, U) = 1 \). Table 7 summarizes these results.

Each tentative diagnosis has counters associated with it: \( d_{00}, d_{01}, d_{10}, d_{11} \). We need a set of counters exponential in the maximum number of module faults we consider. The diagnosis counters are incremented as follows for every \( d \). In a failing itinerary involving a bad module of \( d \), \( d_{11} \) is incremented otherwise \( d_{10} \) is incremented. \( p(\text{good}|d) \) and \( p(\text{bad}|d) \) can now be computed directly in the same way as the single fault intermittent case.

Consider again the system of Figure 5 considering both single and double faults. Initially there are 22 possible diagnoses (1 no faults, 6 single faults, 15 double faults). The first itinerary is \( (A, B, C, D, E, F, \text{Fail}) \). Now \( d_{11} = 1 \) for all remaining 21 diagnoses. Now we see \( (A, B, C, \text{Success}) \). This increments \( d_{10} \) for 17 diagnoses (shortened for brevity sake): \( A, B, C, A, B, A, D, A, A, D, A, E, A, F, B, D, B, E, B, F, C, C, F, D, D, F \). Table 4 lists the counts after repeating the same 4 itineraries as for the single intermittent fault case and Table 5 lists the posterior probabilities for the suspect diagnoses.

Table 4: \( d_{11}, d_{10} \) for multiple intermittent faults.

<table>
<thead>
<tr>
<th>( d )</th>
<th>( d_{11} )</th>
<th>( d_{10} )</th>
<th>candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>4</td>
<td>6</td>
<td>A, B, C, D, E, F</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>2</td>
<td>A, B, C, A, D, A, C, C, D, D, E, F, E, F</td>
</tr>
</tbody>
</table>

Table 5: Posterior probabilities for multiple intermittent faults.

<table>
<thead>
<tr>
<th>( p )</th>
<th>candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, B, C, D, E, F</td>
</tr>
<tr>
<td>0.22</td>
<td>A, B, C, E, F</td>
</tr>
<tr>
<td>0.16</td>
<td>A, B, C, A, D, B, C, D, D, E, F, E</td>
</tr>
<tr>
<td>1.6 \times 10^{-09}</td>
<td>A, B, A, C, A, D, A, B, C, D, C, D, E, D, E, F</td>
</tr>
<tr>
<td>1.2 \times 10^{-09}</td>
<td>A, E, A, F, B, C, E, C, F</td>
</tr>
</tbody>
</table>

The posterior probability of any particular component failed is:

\[
p(C|O, U) = \sum_{n \in S} p(n).
\]

Where \( S \) is the remaining set of suspect candidates (those for which \( m_{01} = 0 \)). Suppose we see the itineraries \( (A, B, C, \text{fail}) \) followed by \( (D, E, F, \text{fail}) \). In the single fault case, this would produce an error. In our example, \( p = 0.09 \) for diagnoses \( A, E, B, E, C, F \) and \( p = 0.16 \) for diagnoses \( A, D, B, D, C, D \). The individual component failure probabilities are: \( p = 0.33 \) for \( C \), \( A \), and \( B \); \( p = 0.26 \) for \( E \) and \( F \); and \( p = 0.48 \) for \( D \). The probabilities sum to 2 because the system contains 2 faults.

7 Replacement

The posterior module probabilities computed in our approach are the probabilities of misdiagnosis: that a replacement of the module will not provide any improvement in system performance. The decision whether to replace a module or not depends on the cost of replacement and that is out of scope of this paper. When a module is replaced, its prior is set using past experience with these types of modules, and all its counters \( m_{ij} \) are set to 0.
Figure 8: Summary of the observation function in the multiple intermittent case for an observation $o$ of itinerary $U$.

8 Capabilities

The catastrophic fault assumption is not correct for complex modules. Each module type has a set of actions it can perform. One of those actions may be faulty, but the module may always succeed at other actions. Therefore, we apply the framework we have developed to actions, not modules. Each capability fails approximately independently. Figure 9 illustrates a 2 way module with 2 capabilities. Figure 10 illustrates a 3 way module with 6 capabilities. Figure 11 illustrates 5 modules of the two types connected together. Circles indicate rollers, triangles indicate sensors, and two sheets of paper are indicated in red. Note that three modules can be acting on the same sheet of paper at one time.

Figure 9: A more detailed figure of a two way module. The 2 possible paper movements (capabilities) are indicated with arrows on the diagram (paper can enter from the right and exit left, or enter from the left and exit right).

It is possible to design machine configurations where a failure in the output capability of one module cannot be distinguished from a failure in the input capability of the connected module. In our framework, this will show up as a double fault when in fact only one of the two modules if faulted. We avoid this confusion by applying an idea from digital circuits to collapse indistinguishable faults. In addition, we always allow multiple faults: we have found most equipment always contains multiple, low frequency, intermittent faults.

9 Initial Results

The acid success test is whether the posterior probabilities calculated by our approach, when incorporated into a larger system, improve overall performance (including planning, diagnosing and production). We ran two experiments: (1) posterior probabilities were assigned randomly, and (2) posterior probabilities computed using the approach of this paper. We presumed single faults and both experiments used the single-fault exoneration rule. The only difference was the posterior probabilities assigned to the unexonerated single faults. We measured the number of sheets needed to isolate the module once a fault is detected. Table 6 lists initial results. The first column is a single fault intermittent rate. The second column is the number of sheets needed using our approach and the third column is the number sheets needed using a random (with exoneration) approach. The table shows our approach requires far fewer wasted sheets to isolate a fault. The description of the overall system and more analysis of performance can be found in [Kuhn et al., 2008].

10 Conclusions

This paper laid out a framework for diagnosing any combination of persistent and intermittent faults in a continuously operating piece of machinery where objects, not signals, are passed from one module to the next. With this extension to model-based diagnosis we have applied on-line diagnosis to modular reprographic equipment. More importantly, it extends model-based diagnosis to the real challenges faced in diagnosing manufacturing plants, packaging equipment, laboratory test equipment, etc.
References


A Framework and Algorithm for Model-Based Active Testing

Alexander Feldman\textsuperscript{1} and Gregory Provan\textsuperscript{2} and Arjan van Gemund\textsuperscript{1}
\textsuperscript{1}Delft University of Technology
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
\textsuperscript{2}University College Cork, College Road, Cork, Ireland
Tel: +353 21 4901816, Fax: +353 21 4274390, e-mail: g.provan@cs.ucc.ie

Abstract

Due to model uncertainty and/or limited observability, the number of possible diagnoses or the associated probability mass distribution may be unacceptable as the basis for important decision-making. In this paper we present a new algorithmic approach, called FRACTAL (FRAmework for ACtive Testing ALgorithms), which, given an initial diagnosis, computes the shortest sequence of additional test vectors that minimizes diagnostic entropy. The approach complements probing and sequential diagnosis (ATPG), applying to systems where only additional tests can be performed by using a subset of the existing system inputs while observing the existing outputs (called "Active Testing"). Our algorithm generates test vectors using a myopic, next-best test vector strategy, using a low-cost approximation of diagnostic information entropy to guide the search. Results on a number of 74XXX/ISCAS85 combinational circuits show that diagnostic certainty can be significantly increased, even when only a fraction of inputs are available for active testing.

1 Introduction

Model-Based Diagnosis (MBD) [de Kleer and Williams, 1987] is an area of abductive inference that uses a system model, together with observations about system behavior to isolate sets of faulty components (diagnoses) that explain the observed behavior. One of the advantages of MBD over related approaches (e.g., simulation-based) is that MBD can cope with arbitrary degree of uncertainty in the system model and in the observation. In the latter case MBD computes \textit{all} or an approximation to all diagnoses. The number of diagnoses can be large, exponential of the number of components in the worst-case.

This ambiguity (uncertainty) of the diagnostic result poses a typical problem to MBD. Due to modeling uncertainty (e.g., weakness due to ignorance of abnormal behavior or need for robustness) and limited number of observations (sensor-lean systems, limited observation horizons), the failure probability mass is distributed over multiple diagnoses. This high information entropy of the diagnostic result makes it difficult for an operator or a reconfiguration (planning) component to decide with sufficient certainty.

Given a set of plausible diagnoses, in certain situations one can devise additional tests that narrow down the ambiguity (reduces the set of diagnoses). When measurements can be made this is a good way to do that [de Kleer and Williams, 1987]. However, in many circumstances there are no provisions for sensing additional variables (e.g., a satellite that cannot be physically reached). In such cases, the only thing that can be done is to actively control (a subset of) inputs, executing a part of the existing system functionality (e.g., invoking built-in test capabilities, or otherwise), the associated observations being used to further narrow down the diagnostic solution space.

Under no constraints, this would mean applying a test vector on \textit{all} inputs such as in sequential diagnosis (and ATPG) where a sequence of tests is applied to target a fault. In many situations, however, this would too much interfere with the system and its environment. Usually, there is a subset of inputs, called control inputs, that can be manipulated by a diagnostic engine to execute tests. This approach is coined "active testing". Loosely speaking, an active testing problem is: given a system model and an initial observation and diagnosis, to compute the set of input test vectors that will minimize diagnostic ambiguity with the least number of test vectors.

In this paper we present a framework, called FRACTAL (FRAmework for ACtive Testing ALgorithms), in which we define active testing and present algorithms to solve the active testing problem. Our contributions are as follows:

- We define the active testing problem and describe various instances of the problem;
- We define diagnostic ambiguity in terms of information entropy and propose a low-cost estimation amenable to active testing;
- We define a stochastic, myopic strategy to solving the active testing problem and outline an algorithm to solve the active testing problem;
- We study the performance of our algorithm on the 74XXX/ISCAS85 combinational benchmark suite.

To the best of our knowledge, this is the first approach to defining and solving the active testing problem, generalizing over sequential diagnosis and ATPG. Furthermore our
method is based on MBD which is beneficial in that very little assumptions about the model and the observations are required. Our results show that controlling a small fraction of the inputs can reduce the number of remaining diagnoses at a small diagnostic cost whereas a reduction of entropy would be impossible for a passive approach. Our method is also computationally efficient as it uses a stochastic approach and is relevant to practice as it can be effectively used to disambiguate faults in complex autonomous systems.

2 Related Work

The problem of sequential diagnosis has received considerable attention in the literature. Our notion of active testing is related to that of Pattipati et al. [Pattipati et al., 1990; Raghavan et al., 1999], except that we compute diagnoses rather than caching all diagnoses in a fault dictionary, we assume all tests have identical costs, and we assume all faults are equally likely, a priori. In addition to that, whereas the test matrix in sequential diagnosis is fixed, we allow part of the inputs to be supplied by the environment in every step of the diagnostic process, which makes our framework more suitable for online fault isolation.

Note that our task is harder than that of Raghavan et al., 1999, since they do diagnosis lookup using a fault dictionary, and still show that the sequential diagnosis task is NP-hard; in our case we compute a new diagnosis after every test. Hence we have an NP-hard sequential problem interleaved with the complexity of diagnostic inference at each step.\footnote{In our case the complexity of diagnostic inference is $\Sigma_2^P$-hard.}

The framework proposed by Pattipati et al. has been extended to an AND/OR-tree technique that is optimal [Kundakcioglu and Unluyurt, 2007]. We note that optimal test sequencing is infeasible for the size of problems in which we are interested.

Rish et al. [Zheng et al., 2005; Brodie et al., 2003] define a similar framework, but cast their models in terms of Bayesian networks. Our notion of entropy is the size of the diagnosis space, whereas Rish et al. use decision-theoretic notions of entropy to guide test selection.

The diagnosis framework that we propose is submodular, in the terms described in [Krause and Guestrin, 2007], i.e., the informativeness of tests exhibits diminishing returns the more tests that we do. In future work we plan to compare our stochastic algorithms to the randomized algorithms that have been developed for submodular functions.

In comparison to all of this work, the main contributions of our paper are:

- A model-based framework for combining multiple-fault and sequential diagnosis and the introduction of reasoning with respect to modifiable/non-modifiable observable variables;
- A characterization of diagnostic entropy in terms of the size of the diagnosis space;
- approximating the size of the diagnosis space in terms of the number of different observations;
- A stochastic algorithm for efficiently estimating the number of different observations and resulting diagnoses.

3 Technical Background

Our discussion starts by adopting the relevant MBD notions [de Kleer and Williams, 1987].

Central to MBD, a model of an artifact is represented as a propositional Wff over a set of variables. We will discern three subsets of these variables: assumable, observable\(^2\) and control variables. This gives us our initial definition:

Definition 1 (Diagnostic System). A diagnostic system $DS$ is defined as the triple $DS = \langle SD, COMPS, OBS \rangle$, where $SD$ is a propositional theory over a set of variables $V$, $COMPS \subseteq V$, $OBS \subseteq V$, $COMPS$ is the set of assumables, and $OBS$ is the set of observables.

Throughout this paper we assume that $OBS \cap COMPS = \emptyset$, and $SD \not\models \bot$. Furthermore, to avoid handling inconsistencies, we restrict $SD$ to models for which $SD \land \alpha \not\models \bot$ for any (possibly partial) assignment $\alpha$ to the variables in $OBS$.

3.1 A Running Example

We will use the Boolean circuit shown in Fig. 1 as a running example for illustrating all notions and the algorithm shown in this paper. The 2-to-4 line demultiplexer consists of four Boolean inverters and four and-gates.

Figure 1: A demultiplexer circuit

The expression $h \Rightarrow (o \leftrightarrow \neg i)$ models an inverter, where the variables $i, o$, and $h$ represent input, output, and control respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \leftrightarrow i_1 \land i_2 \land i_3)$. The above propositional formulae are copied for each gate in Fig. 1 and their variables subscripted and renamed in such a way as to ensure a proper disambiguation and to connect the circuit. The result is the following

\(^2\)In the MBD literature the assumable variables are also referred to as “component”, “failure-mode”, or “health” variables. Observable variables are also called “measurable” variables.
The assumable variables are COMPS = \{h_1, h_2, \ldots, h_8\} and the observables are OBS = \{a, b, i, o_1, o_2, o_3, o_4\}. Note the conventional selection of the sign of the “health” variables \(h_1, h_2, \ldots, h_n\). Other authors use “ab” for abnormal.

3.2 Diagnosis

The traditional query in MBDB computes terms of assumable variables which are explanations for the system description and an observation.

Definition 2 (Diagnosis). Given a system DS, an observation \(\alpha\) over some variables in OBS, and an assignment \(\omega\) to all variables in COMPS, \(\omega\) is a diagnosis iff \(\text{SD} \land \alpha \land \omega \not\models \bot\).

We denote the set of all diagnoses of a model SD and an observation \(\alpha\) as \(\Omega(\text{SD}, \alpha)\) and the number of all diagnoses as \(|\Omega(\text{SD}, \alpha)|\). Continuing our running example, consider an observation vector \(\alpha_1 = \langle a \land \neg b \land i \land o_4\rangle\). There are a total of 256 possible assignments to all variables in COMPS and \(|\Omega(\text{SD}, \alpha_1)| = 200\). Example diagnoses are \(\omega_1 = \langle h_1 \lor h_2 \lor \ldots \lor h_7 \land \neg h_8\rangle\) and \(\omega_2 = \neg h_1 \land h_3 \land h_5 \land h_6 \land h_7 \land h_8\).

As it is typical for underconstrained models to have many diagnoses (exponential to the number of components in the worst case, as in the above, weak, example model), we will impose (partial) ordering on the diagnoses and will consider only diagnoses which satisfy some minimality criterion.

Definition 3 (Cardinality of a Diagnosis). The cardinality of a diagnosis, denoted as \(|\omega|\), is defined as the number of negative literals in \(\omega\).

According to Def. 3, we have \(|\omega_1| = 1\) and \(|\omega_2| = 2\). Next, let us focus on the diagnoses of minimal cardinality.

Definition 4 (Minimal-Cardinality Diagnosis). A diagnosis \(\omega^\leq\) is defined as Minimal-Cardinality (MC) if no diagnosis \(\omega^\leq\) exists such that \(|\omega^\leq| < |\omega^\leq|\).

Other authors use different minimality criteria such as subset-minimality diagnoses, probability-minimal diagnoses, kernel diagnoses (in a slightly different diagnostic framework), etc. [de Kleer et al., 1992]. Our selection of minimality criterion is such that it does not characterize all diagnoses but is often seen in practice due to the prohibitive cost of computing a characterizing set of diagnoses.

Consider an observation vector \(\alpha_2 = \langle a \land \neg b \land i \land \neg o_1 \land o_4\rangle\). There are 6 MC diagnoses of cardinality 2 consistent with \(\text{SD} \land \alpha_2\) and counting these MC diagnoses is a common problem in MBDB.

Definition 5 (Number of Minimal-Cardinality Diagnoses). The number of MC diagnoses of a system DS given an observation \(\alpha\) over some variables in OBS is denoted as \(|\Omega^\leq(\text{SD}, \alpha)|\), where \(\Omega^\leq(\text{SD}, \alpha)\) is the set of all MC diagnoses of \(\text{SD} \land \alpha\).

It is easy to compute the number of MC diagnosis for the circuit in Fig. 1: \(|\Omega^\leq(\text{SD}, \alpha_1)| = 1\) and \(|\Omega^\leq(\text{SD}, \alpha_2)| = 6\).

4 Sequential Diagnosis

Typically, due to uncertainty in the model (e.g., ignorance of abnormal behavior) and in the observation vectors (partial observability), there is more than one MC diagnosis. To reduce this uncertainty and to pinpoint the exact cause of failure, diagnosticians often combine a sequence of diagnostic experiments, where, whenever possible, appropriate input vectors are supplied, generating tests that optimally reduce \(|\Omega|\). If this process of successive application of MBDB in time includes dynamic reconfiguration of the system under test, then we call the process active testing.

Definition 6 (Diagnostic Sequence). Given a system DS, a diagnostic sequence \(S\) is defined as a k-tuple of terms \(S = \langle \alpha_1, \alpha_2, \ldots, \alpha_k\rangle\), where \(\alpha_i (1 \leq i \leq k)\) is an instantiation of the variables in OBS.

The cost of a diagnostic sequence, denoted as \(|S|\), is defined as the number of terms in \(S\) (respectively the number of MBDB experiments performed by a diagnostician).

An important assumption throughout this paper is that the health of the system under test does not change during the test (i.e., intermittent faults are outside the scope of this study).

Assumption 1 (Non-Intermittence). Given an system DS, an actual health state for its components \(\omega_\ast\), and a diagnostic sequence \(S\), we assume that \(\omega_\ast \in \Omega(\text{SD}, \alpha_i)\) for \(1 \leq i \leq |S|\).

It is intuitive that for non-intermittent systems, the diagnostician can combine the results from different application of MBDB to reduce the diagnostic uncertainty.

Lemma 1. Given a system DS, a health state for its components \(\omega_\ast\), and a diagnostic sequence \(S\), it follows that \(|S|\) \(\omega_\ast \in \bigcap_{i=1}^{S} \Omega(\text{SD}, \alpha_i)\).

Proof. The above statement follows immediately from the non-intermittence assumption and Def. 2. □

Obviously, Lemma 1 can be applied only in the cases in which all diagnoses of a model and an observation are considered. If we compute minimal-diagnoses in a weak-fault model, for example (cf. [de Kleer et al., 1992]), the intersection operator has to be redefined to handle subsumptions. The problem with intersecting diagnostic sets worsens if we consider non-characterizing sets of diagnoses (e.g., MC diagnoses or first \(n\) diagnoses). To solve this issue we will provide our own consistency-based intersection operator.

Definition 7 (Consistency-Based Intersection). Given a system description SD, an initial observation \(\alpha\), a (possibly non-characterizing) set of diagnoses \(D\) of \(\text{SD} \land \alpha\), and a posteriori observation \(\alpha'\), the intersection of \(D\) with the diagnoses \(\text{SD} \land \alpha'\), denoted as \(\Gamma_D(D, \alpha')\), is defined as the set \(D' (D' \subseteq D)\) such that for each \(\omega \in D'\) it holds that \(\text{SD} \land \alpha' \land \omega \not\models \bot\).
The intersection operator \( \Omega \cap (D, \alpha) \) refines the set of prior diagnoses \( D \), leaving only diagnoses supported by both observations. It is straightforward to generalize the above definition to a diagnostic sequence \( S \).

**Definition 8** (Remaining Minimal-Cardinality Diagnoses). Given a diagnostic system \( DS \) and a diagnostic sequence \( S \), the set of remaining diagnoses \( \Omega^S \) is defined as \( \Omega^S = \Omega^1 \cdots \Omega^k (\Omega \cap (D, \alpha_1), \alpha_2, \ldots, \alpha_k) \).

It is clear that if we consider the first \( k \) terms of a sequence \( S \) (forming a subsequence \( S^k \)), the size of the set of remaining diagnoses \( |\Omega^S| \) decreases monotonically when increasing \( k \).

Note that we use \( |\Omega^S| \) instead of the more precise diagnostic entropy as defined in [de Kleer and Williams, 1987] and subsequent works. In particular, if all diagnoses of a model and an observation are of minimal-cardinality and the failure probability of each component is the same, then the gain in the diagnostic entropy can be directly computed from \( |\Omega^1| \).

5 An Active Testing Framework

Note that in our MBD use of sequential diagnosis, the observation terms are always determined by “nature” \(^1\). It is often the case, though, that there are inputs (in MBD input and outputs are normally not distinguished and they are both considered as observables) which are not only measurable but also modifiable. We will call these inputs **controls** and we will see that computing values for these control variables can improve the optimality of the diagnostic process.

5.1 Optimal Control

Extending the diagnostic system from Def. 1 and separating the controllable from non-controllable observations gives us the following definition:

**Definition 9** (Active Testing System). An active testing system \( ATS \) is defined as the 4-tuple \( ATS = (SD, COMPS, CTL, OBS) \), where SD is a propositional theory over a set of variables \( V \), \( COMPS \subseteq V \), \( CTL \subseteq V \), \( OBS \subseteq V \), \( COMPS \) is the set of assmumables, \( CTL \) is the set of controls, and \( OBS \) is the set of observables.

Furthermore, although this is not strictly necessary, whenever convenient, we will be splitting the set of observables \( OBS \) into inputs \( IN \) and outputs \( OUT \) (\( OBS = IN \cup OUT, IN \cap OUT = \emptyset \)). Hence, from now on, the observables from the preceding sections will be split into “modifiable” inputs (or controls) \( CTL \), “non-modifiable” inputs \( IN \) and outputs \( OUT \). For the assignments to the inputs, outputs, and controls we will conventionally use (subscripted and superscripted when necessary) \( \alpha, \beta, \) and \( \gamma \), respectively.

Note the distinction between observation terms and control terms. In a typical diagnostic scenario, the observation terms \( (\alpha_1, \alpha_2, \ldots, \alpha_k) \) are determined by “nature”, while the control terms \( (\gamma_1, \gamma_2, \ldots, \gamma_k) \) are set by the diagnostician.

Next, let us consider a diagnostic sequence \( S \) whose terms are split into controls and (non-modifiable) inputs \( (S = (\alpha_1 \land \gamma_1, \alpha_2 \land \gamma_2, \ldots, \alpha_k \land \gamma_k)) \). In such a sequence \( S \), a diagnostician would attempt to minimize the size of the remaining diagnoses \( |\Omega^S| \) by supplying “optimal” \( \gamma_i \) \( (1 \leq i \leq k) \) terms. Ideally, there would be exactly one remaining diagnosis \( \omega_\alpha \) at the end of the sequence. In general, however, there may be more, depending on the model and observability.

**Problem 1** (Optimal Control Input). Given a system ATS, and a sequence \( S = (\alpha_1 \land \gamma_1, \alpha_2 \land \gamma_2, \ldots, \alpha_k) \), where \( \alpha_i \) \( (1 \leq i \leq k) \) are OBS assignments and \( \gamma_1 \) is a CTL assignment, compute a minimal sequence of CTL assignments \( \gamma_2, \ldots, \gamma_k \), such that \( |\Omega^S| \) is minimized.

Problem 1 uses \( \gamma_1 \) because our problem is different from sequential ATPG in the sense that we don’t compute tests for specific target diagnosis \( \omega_\alpha \) (in which case there is no need to have an initial control \( \gamma \) and observation \( \alpha \)). In the active testing problem, the situation is different: we target any health state, so initial observation and control are required.

In this paper we will avoid making assumptions on the values of the observable terms \( \alpha_1, \alpha_2, \ldots, \alpha_k \). For experimenting with active testing algorithms these can be computed from random inputs and the propagation of the injected fault. There is only one special case, however, which is worth distinguishing: \( \alpha_1 = \alpha_2 = \cdots = \alpha_k \) (consider, e.g., a system under test which supplies constant observation because it is stationary, paused, pending an abort or reconfiguration, etc.).

**Problem 2** (Optimal Control Input for a Persistent Input). Given an active testing system ATS, and a sequence \( S = (\alpha \land \gamma_1, \alpha, \ldots, \alpha) \), where \( |S| = k \), \( \alpha \) is an OBS assignment and \( \gamma_1 \) is a CTL assignment, compute a minimal sequence of CTL assignments \( \gamma_2, \ldots, \gamma_k \), such that \( |\Omega^S| \) is minimized.

In practice, a diagnostician does not know what the next observation will be. Fully solving an active testing problem would necessitate the conceptual generation of a tree with all possible observations and associated control assignments in order to choose the sequence that, on average, constitutes the shortest (optimal) path over all possible assignments.

The sequential diagnosis problem studies optimal trees when there is a cost associated with each test [Tu and Patipati, 2003]. When costs are equal, it can be shown that the optimization problem reduces to a next best control problem (assuming one uses information entropy). In this paper a diagnostician who is given a diagnostic session \( S \) and who tries to compute the **next** optimal control assignment would try to minimize the expected number of remaining diagnoses \( |\Omega^S| \).

5.2 Expected Intersection Size

Clearly, \( |\Omega^1| \) is the goal function to be minimized (apart from \( k \)). Next, we will compute the expected number of diagnoses for a set of observable variables \( M \subseteq OBS \).

Note that the initial observation \( \alpha \) and the set of MC diagnoses \( D = \Omega^0 (SD, \alpha) \) modify the probability density function (pdf) of subsequent outputs\(^4\) (observations), i.e., a sub-

\(^1\)Note, that in our presentation “sequential diagnosis” is used in the MBD context, which is slightly different from its original presentation, but still compatible. Normally, sequential diagnosis is the art of finding optimal test sequences where typically all inputs are controllable, and where “nature” is only in charge of computing the outputs. In our case, by “nature” we understand the environment (consider the case in which the system description is embedded within a copier that is paused).

\(^4\)In MBD there is no problem not discerning outputs from observables, “assigning values” to outputs, etc. We leave it to the read-
sequent observation α′ changes its a priori likelihood. The (non-normalized) a posteriori probability of an observation α′, given an MC operator and an initial observation α is:

\[ \Pr(\alpha'|SD, \alpha) = \frac{|\Omega^\gamma(\Omega^\leq(SD, \alpha), \alpha')|}{|\Omega^\leq(SD, \alpha)|} \]  

(1)

The above formula comes by quantifying how a given a priori set of diagnoses restricts the possible outputs (i.e., we take as probability the ratio of the number of remaining diagnoses to the number of initial diagnoses). Note that, in practice, there are many α for which Pr(α′|SD, α) = 0 because a certain fault heavily restricts the possible outputs of a system (i.e., the set of the remaining diagnoses in the denominator is empty).

The expected number of remaining MC diagnoses for a variable set M, given an initial diagnosis α, is then the weighted average of the intersection sizes of all possible instantiations over the variables in M (the weight is the probability of an output):

\[ E^\leq(SD, M|\alpha) = \frac{\sum_{\alpha' \in M^*} |\Omega^\gamma(D, \alpha')| \cdot \Pr(\alpha'|SD, \alpha)}{\sum_{\alpha' \in M^*} \Pr(\alpha'|SD, \alpha)} \]  

(2)

where \( D = \Omega^\leq(SD, \alpha) \) and \( M^* \) is the set of all possible assignment to the variables in \( M \). Replacing (1) in (2) and simplifying gives us the following definition:

**Definition 10 (Expected Minimal-Cardinality Diagnoses Intersection Size).** Given a system ATS and an initial observation \( \alpha \), the expected remaining number of MC diagnoses \( E^\leq(SD, OBS|\alpha) \) is defined as:

\[ E^\leq(SD, OBS|\alpha) = \frac{\sum_{\alpha' \in OBS^*} |\Omega^\gamma(\Omega^\leq(SD, \alpha), \alpha')|}{\sum_{\alpha' \in OBS^*} |\Omega^\gamma(\Omega^\leq(SD, \alpha), \alpha')|} \]

where \( OBS^* \) is the set of all possible assignment to all variables in OBS.

In what follows we will compute the expected number of remaining MC diagnoses.

6 An Algorithm for Active Testing

In this section we will consider algorithms for solving the active testing problem. We start with a description of a naïve, exact, table-based method. The memory and time requirements of this exact method are prohibitive, hence the bulk of this section proposes a more efficient, randomized algorithm.

6.1 Prohibitive Complexity of Exhaustive Search

Consider our running example with an initial observation vector (and control assignment) \( \alpha_3 \land \gamma_3 = a \land b \land i \land \alpha_1 \land \neg\alpha_2 \land \neg\alpha_3 \land \neg\alpha_4 \), where \( \gamma_3 = i \) is chosen as the initial control input. The four MC diagnoses of \( SD \land \alpha_3 \land \gamma_3 \) are \( D_1 = \{\neg h_1, \neg h_3\}, D_2 = \{\neg h_2, \neg h_3\}, D_3 = \{\neg h_4, \neg h_5\}, \) and \( D_6 = \{\neg h_3, \neg h_8\} \).

An exhaustive algorithm would compute the expected number of diagnoses for each of the \( 2^{\text{CTL}} \) next possible control assignments. In our running example we have one control variable \( i \) and two possible control assignments (\( \gamma_5 = i \) and \( \gamma_6 = \neg i \)). To compute the expected number of diagnoses, for each possible control assignment \( \gamma \) and for each possible observation vector \( \alpha \), we have to count the number of initial diagnoses which are consistent with \( \alpha \land \gamma \).

Computing the intersection sizes for our running example gives us Table 1. Note that, in order to save space, Table 1 contains rows for those \( \alpha \land \gamma \) only, for which \( \Pr(\alpha \land \gamma) \neq 0 \), given the initial diagnoses \( D_3 - D_6 \) (and, as a result, \( \Omega^\gamma(\Omega^\leq(SD, \alpha_3 \land \gamma_3), \alpha \land \gamma) \neq 0 \)). It is straightforward to compute the expected number of diagnoses for any control assignment with the help of this marginalization table. In order to do this we have to (1) filter out those lines which are consistent with the control assignment \( \gamma \) and (2) compute the sum and the sum of the squares of the intersection sizes (the rightmost column of Table 1).

| \( i \) | \( a \) | \( b \) | \( \alpha_1 \) | \( \alpha_2 \) | \( \alpha_3 \) | \( \alpha_4 \) | \( \Pr \) | \(|\Omega^\gamma|\) |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| F   | F   | F   | F   | F   | F   | 0.03125 | 1   |
| F   | F   | F   | F   | F   | T   | 0.0625  | 2   |
| F   | F   | T   | F   | F   | F   | 0.03125 | 1   |
| F   | F   | T   | F   | F   | T   | 0.03125 | 1   |
| F   | T   | F   | F   | F   | F   | 0.03125 | 1   |
| T   | F   | F   | F   | F   | T   | 0.0625  | 2   |
| T   | F   | F   | F   | T   | F   | 0.03125 | 1   |
| T   | F   | T   | F   | F   | T   | 0.03125 | 1   |
| T   | T   | F   | F   | F   | T   | 0.0625  | 2   |
| T   | T   | F   | T   | T   | F   | 0.03125 | 1   |

Table 1: Marginalization table for \( SD \) and \( \alpha_3 \)

To compute \( E(SD, OBS|\alpha_3 \land \neg i) \), for example, we have to find the sum and the sum of the squares of the intersection sizes of all rows in Table 1 for which column \( i \) is \( F \). It can be checked that \( E(SD, OBS|\alpha_3, \neg i) = 24/16 = 1.5 \). Similarly, \( E(SD, OBS|\alpha_3 \land i) = 34/16 = 2.125 \). Hence an optimal diagnostician would consider a second measurement with control setting \( \gamma = i \).

The obvious problem with the above brute-force approach is that the size of the marginalization table is, in the worst-case, exponential in \( |OBS| \). Although many of the rows in the marginalization table can be skipped as the intersections are empty (there are no consistent prior diagnoses with the respective observation vector and control assignment), the construction of this table is computationally so demanding that
we will consider an approximation algorithm (to construct Table 1 for our tiny example, the exhaustive approach had to perform a total of 512 consistency checks).

6.2 Approximation of the Expectation

Our algorithm for active testing consists of (1) a randomized algorithm for approximating the expected number of remaining diagnoses and (2) a greedy algorithm for searching the space of control assignments. We continue our discussion with approximating the expectation.

The key insight which allows us to build a faster method for computing the expected number of remaining diagnoses is that the prior observation (and respectively a set of MC diagnoses) shifts the probability of the outputs. Hence, an algorithm which samples the possible input assignments (it is safe to assume that inputs are equally likely) and counts the number of different observations given the set of prior diagnoses would produce a good approximation.

Algorithm 1 Approximate expectation

1: function EXPECTATION(ATS, γ, D) returns a real
2: inputs: ATS, active testing system
3: γ, term, system configuration
4: D, set of diagnoses, prior diagnoses
5: local variables: α, β, ω, terms
6: s, q, integers, initially 0
7: S, a set of terms, samples
8: E, a real, expectation
9: 2: S ← ∅
10: repeat
11: 4: α ← RANDOMINPUTS(SD, IN)
12: 5: for all ω ∈ D do
13: 6: β ← INFEROUTPUTS(SD, OUT, α ∧ γ, ω)
14: 7: if α ∧ β ̸∈ S then
15: 8: S ← S ∪ {α ∧ β}
16: 9: s ← s + |Yα∧β(D, α ∧ β ∧ γ)|
17: 10: q ← q + |Yα∧β(D, α ∧ β ∧ γ)|
18: 11: E ← q/s
19: end if
20: end for
21: until TERMINATE(E)
22: return E

Algorithm 1 uses a couple of auxiliary functions: RANDOMINPUTS assigns random values to all inputs and INFEROUTPUTS computes all outputs from the system model, all inputs and a diagnosis.\(^5\) The computation of the intersection size \(|\Omega^\gamma\{D, \alpha \land \beta \land \gamma\}|\) can be implemented in a straightforward manner. It is enough to count those \(\omega \in D\) for which \(S \land \alpha \land \beta \land \gamma \land \omega \neq \perp\).

It can be seen that the value of the expected number of diagnoses \(\hat{E}\) approaches the exact value \(E\) when increasing the number of samples \(|S|\). In particular, \(\hat{E}\) is the exact number of the expected number of diagnoses when all possible input values are considered (in the latter case Alg. 1 simply builds the marginalization table for a given control assignment \(\gamma\)).

Figure 2 shows two examples of \(\hat{E}\) approaching \(E\) for two bigger models (cf. Sec. 7).

The algorithm terminates when a suitable termination criterion (checked by the TERMINATE function) is satisfied. In our implementation TERMINATE returns success when the last \(n\) iterations (where \(n\) is a small constant) leave \(\hat{E}\) unchanged.

6.3 Greedy Control Setting Algorithm

In addition to the approximation for the expectation of the number of diagnoses, we need a faster method for searching the space of possible control assignments (the brute-force approach considers \(2^{\lvert \text{CTL} \rvert}\) control assignments). We will assume that the control literals are independent, flip them one at a time, and accept the new control assignment if it decreases the expected number of remaining MC diagnoses. The approach is shown in Alg. 2, which computes a control assignment for a given active testing system and a prior observation.

Algorithm 2 Optimal next control input

1: function CONTROL(ATS, α) returns a control term
2: inputs: ATS, active testing system
3: α, term, initial observation
4: local variables: γ, γ’, terms, control configurations
5: \(\hat{E}, E',\) reals, expectations
6: D, set of terms, diagnoses
7: \(l, \) literal, control literal
8: 2: D ← Ω\(^\alpha\)(SD, α)
9: 3: \(\hat{E} ←\) EXPECTATION(ATS, γ, D)
10: 4: for all \(l \in \gamma\) do
11: 5: \(\gamma' ← \) FLIPLITERAL(γ, l)
12: 6: \(E' ← \) EXPECTATION(ATS, γ’, D)
13: 7: if \(E' < E\) then
14: 8: \(γ ← γ'\)
15: 9: \(E ← E'\)
16: 10: end if
17: 11: end for
18: 12: return γ
19: 13: end function

\(^5\)This is not always possible in the general case. In our framework, we have a number of assumptions, i.e., a weak-fault model, well-formed circuit, etc. The complexity of INFEROUTPUT varies on the framework and the assumptions.
The above algorithm computes a control assignment minimizing the expected intersection size, given an active testing system and an initial observation (and control assignment). The set of initial diagnoses is computed from the initial observation in line 2. In line 5, Alg. 2 “flips” the next literal in the current control assignment. The auxiliary FLIPLITERAL subroutine simply changes the sign of a specified literal in a term. After each “flip” the expected intersection size is computed with a call to EXPECTATION (cf. Alg. 1). If the new expected intersection size is smaller than the current one, then the proposed control assignment is accepted as the current control assignment and the search continues from there.

The advantage of the greedy approach is that the number of computations of expected number of diagnoses is linear of the number of literals in the control assignment. This is done at the price of some optimality (i.e., the effect of combinations of controls is neglected). It is straightforward to turn Alg. 2 into a full heuristic search.

7 Experimental Results

Next we discuss an implementation of FRACTAL.

7.1 Experimental Framework for Active Testing

We have implemented a FRACTAL experimental framework. The idea is to (1) inject a random fault and then to (2) simulate a manifestation of this fault. Given this initial manifestation we invoke the active testing algorithm for (3) computing an optimal next control setting. After a control setting is generated, the simulator generates (4) another manifestation of the same fault. This allows FRACTAL to (5) refine the set of diagnoses by intersecting them (cf. Def. 7). Steps 3, 4, and 5 are repeated until some termination criterion is satisfied (e.g., the set of diagnoses remains a singleton\(^6\)).

Algorithm 3 uses the same auxiliary functions RANDOMINPUTS and INFEROUTPUTS as in Sec. 6.2. The subroutine RANDOMCONTROLS is similar to RANDOMINPUTS except that it generates assignments to the variables in CTL instead of the ones in OBS. Similarly, RANDOMFAULT generates a random assignment to the assumable variables in COMPIS.

Algorithm 3 maintains a set of remaining diagnoses in  2  which are iteratively refined in line 12. The algorithm terminates when a suitable termination criterion (checked by the TERMINATE function) is satisfied. In our implementation TERMINATE returns success when there is only one remaining diagnosis in D or when the last \(n\) iterations (where \(n\) is a small constant) leave the size of D unchanged. Note that depending on the observability of the model (the contents of OBS), it may never happen that D is reduced to a single diagnosis.

7.2 Implementation Notes and Test Set Description

We have implemented FRACTAL in approximately 1500 lines of C code (excluding minimal-diagnosis code and consistency checking) and it is a part of the LYDIA package.\(^3\)

\(^3\)LYDIA is downloadable from http://fdir.org/lydia/.

Traditionally, MBD algorithms have been tested on diagnostic models of digital circuits like the ones included in the ISCAS85 benchmark suite [Brglez and Fujiwara, 1985]. As models derived from the ISCAS85 circuits are computationally intensive (from a diagnostic perspective), we have also considered four medium-sized circuits from the 74XXX family [Hansen et al., 1999].

All time measurements in this paper are performed on a host with 1.86 GHz Pentium M CPU and 2 Gb of RAM.

7.3 Performance and Optimality of Active Testing

In this experiment we compare the results of FRACTAL to a setting where all inputs are non-modifiable and the initial observation is repeated at every step of the sequence (cf. Sec. 5). Obviously, in the latter case, the initial number of diagnoses can not be reduced any further. The result is shown in the second column of Table 3.

The third column of Table 3 shows the remaining number of diagnoses after \(|S|\) steps (column 4 of Table 3). Finally,
Table 3: Comparison of number of diagnoses with persistent \( \alpha \) to active testing (\(|\text{CTL}| = |\text{IN}|\))

| Name | \( \Omega^L \) | \( \Omega^H \) | \( |S| \) | \( T [s] \) |
|------|--------------|--------------|--------|----------|
| 74182 | 4            | 2            | 4      | 0.6      |
| 74L85 | 8            | 2            | 5      | 2.2      |
| 74283 | 5            | 3            | 4      | 1.5      |
| 74181 | 10           | 1            | 2      | 0.3      |
| c432  | 10           | 1            | 2      | 20.6     |
| c499  | 4            | 4            | 4      | 27.6     |
| c880  | 39           | 8            | 4      | 443.6    |
| c1355 | 5            | 4            | 4      | 104.1    |

Table 4: Minimal expected entropy \( \bar{E} \) for varying controllability (\( c = |\text{CTL}|/(|\text{CTL}| + |\text{IN}|)\))

| Name | \( |\text{CTL}| \) | \( c = 0.25 \) | \( c = 0.5 \) | \( c = 0.75 \) | \( c = 1 \) |
|------|-----------------|---------------|---------------|---------------|---------------|
| 74182 | 4               | 3.26          | 2.36          | 2.25          | 2             |
| 74L85 | 8               | 4.41          | 4.29          | 3.63          | 3             |
| 74283 | 5               | 3.17          | 2.25          | 2.2           | 2.2           |
| 74181 | 10              | 6.09          | 5.28          | 5.18          | 3.8           |
| c432  | 10              | 3.08          | 2.93          | 2.55          | 2.4           |
| c499  | 4               | 4             | 4             | 4             | 4             |
| c880  | 39              | 21.68         | 16.44         | 15.97         | 12.23         |
| c1355 | 5               | 4.06          | 3.4           | 3.4           | 3.4           |

Table 7.4: Minimal Expected Intersection Size

In the experiment that follows we will experiment with computing the expected number of minimal diagnoses with an initial observation only (as opposed to a longer sequence of observations and controls). The result (shown in Table 4) gives an indication on the effect of the control variables on the expected number of remaining diagnoses.

We have seeded our experiments with arbitrary double faults (or single faults for circuits larger than \( C_{1355} \) for faster testing, called \( \text{FRAC-TAL} \). The algorithm consists of a sampling-based method for approximating the entropy and a greedy method for searching the next optimal control setting.

We have implemented the algorithm and experimented on a range of combinational benchmarks. Experiments show that controlling a small fraction of the inputs can reduce the diagnostic uncertainty while minimizing the diagnostic cost.

We argue that active testing can be of broad practical significance, as it can reduce diagnostic uncertainty in situations in which \( \text{MBD} \) alone is not capable of determining exact cause of failure.

As a future work we plan to bound the error of the randomized algorithms and to perform more experiments on additional set of models and observation vectors. We also plan to study the problems complexity and to improve and assess the performance of our implementations.

References


FlexDx: A Reconfigurable Diagnosis Framework

Fredrik Heintz†, Mattias Krysander∗, Jacob Roll∗, Erik Frisk∗
† Dept. of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden
Email: frehe@ida.liu.se
∗ Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
Email: {matkr,roll,frisk}@isy.liu.se

Abstract

Detecting and isolating multiple faults is a computationally intense task which typically consists of computing a set of tests, and then computing the diagnoses based on the test results. This paper describes FlexDx, a reconfigurable diagnosis framework which reduces the computational burden by only running the tests that are currently needed. The method selects tests such that the isolation performance of the diagnostic system is maintained. Special attention is given to the practical issues introduced by a reconfigurable diagnosis framework such as FlexDx. For example, tests are added and removed dynamically, tests are partially performed on historic data, and synchronous and asynchronous processing are combined. To handle these issues FlexDx uses DyKnow, a stream-based knowledge processing middleware framework. The approach is exemplified on a relatively small dynamical system, which still illustrates the computational gain with the proposed approach.¹

1 Introduction

Detection and isolation of multiple faults in a dynamic process is a computationally expensive task, and the cost increases rapidly with the number of faults and the model complexity. A real-time, model-based diagnosis system that supervises a dynamic system with non-linear behavior often consists of a set of precompiled diagnostic tests together with a fault isolation module [3; 14]. The diagnostic tests are based on a formal description of the process, often in the form of differential or difference equations. For this type of system, pre-compiled test is an attractive solution compared to e.g. solutions based on propagating values like GDE.

The computational complexity of such a diagnosis system mainly originates from two sources: complexity of the process model and the number of behavioral modes that are considered. A high capability of distinguishing between faults, especially when multiple faults are considered, requires a large number of diagnostic tests [9]. Also, the more complex the process model is, the more computationally intense is the task of executing the diagnostic tests. In this paper we develop a reconfiguration scheme to handle computational issues while still being able to handle multiple faults. A related approach is presented in [17] although the models and diagnosis techniques are different. Recently, works on on-line reconfiguration of the diagnosis system have appeared. For a related work, see e.g. [2], where Kalman-filters are reconfigured based on diagnosis decisions.

The main idea of this work is to utilize the observation that all tests are not needed at all times, which can be used to reduce the overall computational burden. For example, when starting a fault free system, there is no need to run tests that are designed with the sole purpose of distinguishing between faults. In such a case, only tests that are able to detect faults are needed, which may be significantly fewer compared to the complete set of tests. When a test triggers an alarm and a fault is detected, appropriate tests are started to make it possible to compute a refined diagnosis decision. Such an approach requires a flexible and reconfigurable framework where tests can be added and removed on-line in a controlled fashion, and also be run on historical data.

The objective of this paper is to illustrate how such a dynamic approach to diagnosis can be designed and implemented using linear dynamical process models. In particular, the implementation issues introduced by a reconfigurable diagnosis framework are discussed and a solution using DyKnow [7; 8], a stream-based knowledge processing middleware framework, is described. It will also be shown how such an approach requires controlled ways of initializing the dynamic diagnostic tests, and how to select the new tests to be started when a set of diagnostic tests has generated an alarm.

The reconfigurable diagnosis framework proposed in this paper, named FlexDx, is introduced in Section 2, and the theoretical diagnosis background needed is presented in Section 3. Methods how to determine, in a specific situation, which tests should be started next are treated in Section 4. A proper initialization procedure for dynamic tests is described in Section 5. The complete approach is exemplified on a small dynamic system in Section 6, which, in spite of the relatively small size of the example, clearly illustrates the complexity of the problem and the possible computational gain with the proposed approach. The software framework which facilitates the implementation of FlexDx, DyKnow, is briefly described in Section 7, and finally a summary is given in Section 8.

¹This work is partially supported by grants from the Swedish Aeronautics Research Council (NFFP4-S4203) and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII.
2 FlexDx: A Reconfigurable Diagnosis Framework

As mentioned in the introduction, a framework like FlexDx must be capable of adding and removing tests dynamically while refining the set of diagnoses. This is done in an iterative manner by the following procedure:

1. Initiate the set of diagnoses.
2. Based on the set of diagnoses, compute the set of tests to be performed.
3. Compute the initial state of the selected tests.
4. Run the tests until an alarm is triggered.
5. Compute the current set of diagnoses based on the test results, then go to step 2.

When dealing with multiple fault diagnosis, it has been shown useful to represent all diagnoses with the minimal diagnosis [5]. This representation will also be used here. When FlexDx is started, there are no conflicts and the only minimal diagnosis is the no-fault mode NF, i.e. the set of minimal diagnoses $D$ is set to $\{\text{NF}\}$ in step 1. Step 2 uses a function that given a set of diagnoses $D$ returns the set of tests $T$ to be performed to monitor whether a fault has occurred or to further explore the possible diagnoses. Step 3 initiates each of the tests in $T$. A test includes a residual generator given in state-space form. This means that the start-up of such a residual generator involves the estimation of its initial condition. In step 4, the tests are performed until at least one triggers an alarm and a test result is generated in the form of a set of conflicts [4; 16]. Step 5 computes the new set of diagnoses $D$, given the previous set of diagnoses and the generated set of conflicts. This step can be performed by algorithms handling multiple fault diagnoses [4; 11].

Step 4 and 5 are standard steps used in diagnosis systems and will not be described in further detail. Step 2 and 3 are new steps, needed for dynamically changing the test set and will not be described in further detail. Step 2 and 3 are formed by algorithms handling multiple fault diagnoses [4; 11].

To implement an instance of the FlexDx framework, a number of issues have to be managed besides implementing the algorithms and integrating them to a system. When a potential fault is detected, FlexDx computes the last known fault free time $t_f$ and the new set of residual generators to be started at time $t_f$. To implement this, three issues have to be solved. First, the FlexDx instance must be reconfigured to replace the set of residual generators and their monitors. Second, the computation of the residuals must begin at time $t_f$ which will be in the past. Third, at the same time as FlexDx is computing residuals and performing tests on the historic data, system observations will keep coming at their normal rate. How these issues are solved is described in Section 7.

3 Theoretical Background

The diagnosis systems considered in this paper consist of a set of tests. Each test consists of a residual $r(t)$ that is thresholded such that it triggers an alarm if $|r(t)| > 1$. Note that the threshold can be set to one without loss of generality. It is assumed that the residuals are normalized such that a given false alarm probability $p_{FA}$ is obtained, i.e.

$$P(|r(t)| > 1 | \text{NF}) = p_{FA}$$

The residuals are designed using a model of the process to be diagnosed.

3.1 The Model

The model class considered here is linear differential-algebraic models. It is worth noting that even if the presentation here in the paper relies on results for linear systems, the basic idea is equally applicable also to non-linear model descriptions.

There are several ways to formulate differential-algebraic models. Here, a polynomial approach is adopted, but any model description is possible, e.g. standard state-space or descriptor models. The model is given by the expression

$$H(q)x + L(q)w + F(q)f = V(q)v$$

where $x(t) \in \mathbb{R}^{n_x}$, $w(t) \in \mathbb{R}^{n_w}$, $f(t) \in \mathbb{R}^{n_f}$, and $r(t) \in \mathbb{R}^{n_r}$. The matrices $H(q)$, $L(q)$, $F(q)$, and $V(q)$ are polynomial matrices in the time-shift operator $q$. The vector $v$ contains all unknown signals, which include internal system states and unknown inputs. The vector $w$ contains all known signals such as control signals and measured signals, the vector $f$ contains the fault-signals, and the vector $r$ is white, possibly multidimensional, zero mean, unit covariance Gaussian distributed noise.

To guarantee that the model is well formed, it is assumed that the polynomial matrix $[H(z) \ L(z)]$ has full column rank for some $z \in \mathbb{C}$. This assumption assures that for any noise realization $r(t)$ and any fault signal $f(t)$ there exists a solution to the model equations (2).

3.2 Residual Generation

Residuals are used both to detect and isolate faults. This task can be formulated in a hypothesis testing setting. For this, let $f_i$ denote both the fault signal and the corresponding behavioral mode of a single fault. Let $\mathcal{F}$ be the set of faults.

A pair of hypotheses associated with a residual can then be stated as

$$H_0 : f_i = 0 \text{ for all } f_i \in \mathcal{F}_0$$
$$H_1 : f_i \neq 0 \text{ for some } f_i \in \mathcal{F}_0$$

where $\mathcal{F}_0 \subseteq \mathcal{F}$ is the set of faults the residual is designed to detect. This means that the residual is not supposed to detect all faults, only the faults in $\mathcal{F}_0$. By generating a set of such residuals, each sensitive to different subsets $\mathcal{F}_0$ of faults, fault isolation is possible. This isolation procedure is briefly described in Section 3.3.

In the literature there exists several different ways to formally introduce residuals. In this paper an adapted version of the innovation filter defined in [10] is used. For this, it will be convenient to consider the nominal model under a specific hypothesis. The nominal model under hypothesis $H_0$ above is given by (2) with $V(q) = 0$ and $f_i = 0$ for all $f_i \in \mathcal{F}_0$. With this notion, a nominal residual generator is a linear time-invariant filter $r = R(q)w$ where for all observations $w$, consistent with the nominal model (2) under hypothesis $H_0$, it holds that $\lim_{t \to \infty} r(t) = 0$.

Now, consider again the stochastic model (2) where it is clear that a residual generated with a nominal residual generator will be subject to a noise component from the process noise $v$. A nominal residual generator under $H_0$ is then said
to be a residual generator for the stochastic model (2) if the noise component in the residual \( r \) is white Gaussian noise.

It can be shown [6] that all residual generators \( R(q) \), as defined above, for the stochastic model (2) can be written as

\[
R(q) = Q(q)L(q)
\]

where the matrix operator \( Q(q) \) satisfies the condition \( Q(q)H(q) = 0 \). This means that the residual is computed by \( r = Q(q)L(q)w \) and it is immediate that the internal form of the residual is given by

\[
r = Q(q)L(q)w = -Q(q)F(q)f + Q(q)V(q)v
\]

(3)

Thus, the fault sensitivity is given by

\[
r = -Q(q)F(q)f
\]

(4)

and the statistical properties of the residual under \( H_0 \) by

\[
r = Q(q)V(q)v
\]

(5)

A complete design procedure is given in e.g. [10] for state-space models and in [6] for models on the form (2). The objective here is not to describe a full design procedure, but it is straightforward to verify that the minimal diagnoses in this case are

\[
B = \{ f_1, f_3 \}
\]

and that \( r \) is sensitive to the faults with non-zero transfer function, i.e. how the set of tests \( D \) is minimal. There are many possible ways how this can be done. The method that will be described here is based on the deterministic properties of (2) only and relies on basic principles in consistency-based diagnosis.

A fundamental task in consistency-based diagnosis is to compute the set of consistent modes [4] given a model, a set of possible behavioral modes, and observations. The design goal of the test selection algorithm will be to perform tests such that the set of consistent modes is equal to the set of diagnoses computed by the diagnosis system.

### 4.1 Consistent Behavioral Modes

The deterministic behavior in a behavioral mode \( b \) is described by (2) when \( v = 0 \) and \( f_i = 0 \) for all \( f_j \not\in b \), and the set of observations consistent with \( b \) is consequently given by

\[
O(b) = \{ w | \exists x \exists f_3 (\forall j : f_j \not\in b \rightarrow f_j = 0) \land H(q)x + L(q)w + F(q)f = 0 \}
\]

(6)

This means that a mode \( b \) is consistent with the deterministic part of model (2) and an observation \( w \) if \( w \in O(b) \). Hence, to achieve the goal the set of diagnoses should, given an observation \( w \), be equal to \( \{ b \in B \mid w \in O(b) \} \) where \( B \) denotes the set of all behavioral modes. As mentioned in Section 2, we will use minimal diagnoses to represent all diagnoses. This is possible since (6) implies that \( O(b') \subseteq O(b) \) if \( b' \subseteq b \). Hence, if \( b' \) is consistent it follows that \( b \) is consistent and therefore it is sufficient to check if the minimal consistent modes remain consistent when new observations are processed.

### 4.2 Tests for Checking Model Consistency

Next, we will describe how tests can be used to detect if \( w \not\in O(b) \). Let \( T \) be the set of all available tests and let \( r_1 = Q_1(q)L(q)w \) be the residual corresponding to test \( t_1 \).

A residual generator checks the consistency of a part of the complete model. To determine which part, only the deterministic model needs to be considered. It can be shown [12] that residual \( r_1 \) checks the consistency of \( \xi(q)w = 0 \) where \( \xi(q) \) is a polynomial in the time-shift operator \( q \). By defining the set of consistent observations for tests in a similar way as for models, we define

\[
O(t_1) = \{ w | \xi(q)w = 0 \}
\]

(7)

Now, we can characterize all test sets \( T \) that are capable of detecting any inconsistency between an observation \( w \) and the assumption that \( w \in O(b) \). For this purpose, only tests \( t_i \) with the property that \( O(b) \subseteq O(t_i) \) can be used. For such a test, an alarm implies that \( w \not\in O(t_i) \) which further implies that \( w \not\in O(b) \). This means that a test set \( T \) is capable of detecting any inconsistency of \( w \in O(b) \) if and only if

\[
O(b) = \bigcap_{t_i \in T \setminus O(b) \subseteq O(t_i)} O(t)
\]

(8)

A trivial solution to (8) is \( T = \{ t \} \) where \( O(t) = O(b) \).

### 4.3 The Set of All Available Tests

If \( T \) is not capable of checking the consistency of \( b \), then no subset of tests will be capable of doing this either. Hence, this approach sets requirements on the entire set of tests \( T \). If such set of tests is difficult to obtain for a particular model,
any set of tests will do. By applying the approach to a model consisting of the considered set of tests, a diagnosis system with the same diagnosis capability as the considered set of tests will be the result. In this paper, we will use two different types of test sets $T$ fulfilling (8) for all modes $b \in B$. These are introduced by the following example.

**Example 2** Consider the model

$$
\begin{align*}
    x_1(t + 1) &= \alpha x_1(t) + w_1(t) + f_1(t) \\
    x_2(t) &= x_1(t) + f_2(t) \\
    w_2(t) &= x_2(t) + f_3(t) \\
    w_3(t) &= x_2(t) + f_4(t)
\end{align*}
$$

where $x_i$ are unknowns, $w_i$, known variables, $\alpha$ a known parameter, and $f_i$ the faults. There are $2^3$ modes and the set of observations consistent with each mode is

$$
O(\emptyset) = \{ w \mid w_1(t) + \alpha w_2(t) - w_2(t + 1) = 0 \}
$$

$$
O(\{ f_1 \}) = \{ w \mid -w_2(t) + w_3(t) = 0 \}
$$

$$
O(\{ f_2 \}) = O(\{ f_3 \}) = O(\{ f_4 \}) =
$$

$$
= \{ w \mid w_1(t) + \alpha w_2(t) - w_2(t + 1) = 0 \}
$$

$$
O(\{ f_1, f_2 \}) = \{ w \mid w_1(t) + \alpha w_2(t) - w_2(t + 1) = 0 \}
$$

The behavioral models for the 10 remaining modes $b$ do not contain any redundancy and the observations are therefore not restricted, i.e., $O(b) = \mathbb{R}^3$. In contrast to (6), the sets of consistent observations are here expressed in the same form as for tests, that is with linear differential equations in the known variables only. Any set described as in (6) can be written in this form [15].

The first type of test set $T_1$ will be to design one test for each distinct behavioral model containing redundancy, i.e., for the example $T_1$ consists of four tests $t_i$ such that $O(t_1) = O(\emptyset), O(t_2) = O(\{ f_1 \}), O(t_3) = O(\{ f_2 \}),$ and $O(t_4) = O(\{ f_3 \})$. To check the consistency of $w \in O(\emptyset)$, two linear residuals are needed, which is the degree of redundancy of a model. These two residuals can be combined in a positive definite quadratic form to obtain a scalar test quantity. When stochastic properties are considered, the quadratic form is chosen such that the test quantity conforms to a $\chi^2$-distribution.

Tests for models with a high degree of redundancy can be complex, and the second type of test set $T_2$ includes only the tests for the behavioral models with degree of redundancy 1. For the example, $T_2 = \{ t_2, t_3, t_4 \}$ and by noting that $O(\emptyset) = O(t_i) \cap O(t_j)$ for any $i \neq j$ where $i, j \in \{ 2, 3, 4 \}$, any two tests can be used to check the consistency of $w \in O(\emptyset)$. In [9] it has been shown under some general conditions that $T_2$ fulfills (8) for all modes $b \in B$.

### 4.4 Test Selection Methods

We will exemplify methods that given a set of minimal diagnoses $D$ select a test set $T \subseteq T$ such that (8) is fulfilled for all $b \in D$. An optional requirement that might be desirable is to select such a test set $T$ with minimum cardinality. The reason for not requiring minimum cardinality is that the computational complexity of computing a minimum cardinality solution is generally much higher than to find any solution.

A straightforward method is to use the first type of tests and not require minimum cardinality solutions. Since this type of test set includes a trivial test $O(t_i) = O(b)$ for all modes $b$ with model redundancy, it follows that a strategy is to start the tests corresponding to the minimal diagnoses in $D$.

**Example 3** Consider Example 2 and assume that the set of minimal diagnoses is $D = \{ \emptyset \}$. Then it is sufficient to perform test $t_1$, i.e., $T = \{ t_1 \}$. If the set of minimal diagnoses are $D = \{ \{ f_2 \}, \{ f_3 \}, \{ f_4 \} \}$, then $t_3$ is used to check the consistency of both $\{ f_2 \}$ and $\{ f_4 \}$ and the total set of tests is $T = \{ t_3, t_4 \}$. For this example, this strategy produces the minimum cardinality solutions, but this is not true in general.

A second method is to use the second type of tests and for example require a minimum cardinality solution. The discussion of the method will be given in Section 6 where this method has been applied to a larger example.

### 5 Initialization

When a new test selection has been made, new tests have to be initialized. Since information about faults sometimes are only visible in the residuals for a short time-period after a fault occurrence, we would like a new test to start running before the currently considered fault occurred; otherwise valuable information would be missed. It is also important that the state of the new test gets properly initialized, such that the fault sensitivity is appropriate already from the start, and the residuals can deliver test results immediately. Therefore, the initialization following a new test selection consists of:

1. Estimate the time of the fault from the alarming test(s).
2. Estimate the initial condition for each new test.

Both these steps require the use of historical data, which therefore have to be stored. The fault time estimation will use the historical residuals from the triggered test, while the initial condition estimation uses the measured data from the process before the fault occurred.

#### 5.1 Estimating the Fault Time

There are many possibilities to estimate the fault time. See for example [13; 1] for standard approaches based on likelihood ratios. Here, a window-based test has been chosen. It should be noted, however, that for the given framework, what is important is not really to find the exact fault time, but rather to find a time-point before the fault has occurred. The estimated time-point will be denoted by $t_f$.

Given a number of residuals from an alarming test, $r(1), \ldots, r(k)$, let us compute the sum of the squared residuals over a sliding window, i.e.,

$$
S(t) = \frac{1}{\sigma^2} \sum_{j=1}^{\ell} r^2(t + j), \quad t = 0, \ldots, k - \ell
$$

If the residual generator is designed such that, under the null hypothesis that no fault has occurred, $\{ r(j) \}_{j=1}^{k}$ are white and Gaussian with variance $\sigma^2$, then $S(t) \sim \chi^2(\ell)$ in the fault free case. Hence, $S(t)$ can be used to test whether this null hypothesis has been rejected at different time-points, by a simple $\chi^2$-test. Since it is preferable to get an estimated time-point that occurs before the actual fault time, rather than after, the threshold of the $\chi^2$-test should be chosen such that the null hypothesis is fairly easily rejected. The estimate $t_f$ is
then set to the time-point of the last non-rejected test. Also, in order not to risk a too late estimate, the time-point at the beginning of the sliding window is used.

5.2 Estimating the Initial Condition
Having found \( t_f \), the next step is to initialize the state of the new residual generator. The method used here considers a time-window of samples of \( w(t_f - k), \ldots , w(t_f) \) as input to find a good initial state \( x(t_f) \) of the filter at the last time point of the window.

Consider the following residual generator:

\[
\begin{align*}
x(t + 1) &= Ax(t) + Bu(t) \\
r(t) &= Cx(t) + Dw(t)
\end{align*}
\]

(11) (12)

Assume that \( w(t) = w_0(t) + Nv(t) \) where \( w_0(t) \) is the noise-free data (inputs and outputs) from the process model and \( v(t) \) is Gaussian noise. In fault-free operation, there is a state sequence \( x_0(t) \), such that the output \( r(t) = 0 \) if \( v(t) = 0 \),

\[
x_0(t + 1) = Ax_0(t) + Bu_0(t) \\
0 &= Cx_0(t) + Dw_0(t)
\]

(13) (14)

Given \( w(t) \), \( t = t_f - k, \ldots , t_f \), we would like to estimate \( x_0(t_f) \). This will be done by first estimating \( x_0(t_f - k) \).

From (13) and \( w(t) = w_0(t) + Nv(t) \) we get

\[
0 = R_x x_0(t_f - k) + R_W w_0(t) \\
\Leftrightarrow R_x x_0(t_f - k) + R_W W = R_W D V
\]

(15)

where

\[
R_x = \begin{bmatrix} C & C A \\ C A^T & C A^T C A \end{bmatrix}, \quad R_W = \begin{bmatrix} D & 0 & 0 & \ldots \\ C B & D & 0 & \ldots \\ \vdots & \vdots & \ddots & \ddots \\ C A^k B & \ldots & D \end{bmatrix}
\]

\[
W = \begin{bmatrix} w(t_f - k) \\ w(t_f) \end{bmatrix}, \quad W_0 = \begin{bmatrix} w_0(t_f) \\ w(t_f) \end{bmatrix}
\]

\[
V = \begin{bmatrix} v(t_f - k) \\ v(t_f) \end{bmatrix}, \quad D_V = \begin{bmatrix} N & 0 & \ldots & 0 \\ 0 & N & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & N \end{bmatrix}
\]

Assuming that the distribution of \( V \) is known, \( V \sim N(0, \Sigma_V) \), \( (15) \) means that \( R_x x_0(t_f - k) + R_W W \) is a zero-mean stochastic vector with covariance matrix \( R_x D_V \Sigma_V D_V^T R_x \). Note that the expression above corresponds to the actual residuals obtained when starting in \( x_0(t_f - k) \). Due to the design of the residual generator giving white residuals, this means that \( R_x D_V \Sigma_V D_V^T R_x \approx \sigma^2 I \).

Hence, a reasonable estimate of \( x_0(t_f - k) \) is given by the regular least-squares estimate,

\[
\hat{x}_0(t_f - k) = -(R_x^T R_x)^{-1} R_x^T R_x W
\]

(16)

From this, \( \hat{x}_0(t_f) \) can be computed as

\[
\hat{x}_0(t_f) = A^k \hat{x}_0(t_f - k) + \begin{bmatrix} A^k B & A^{k-2} B & \ldots & AB & B \end{bmatrix} W
\]

The choice of \( k \) is made in advance, based on the computed variance of the initial residuals given \( x_0(t_f) \). The larger \( k \) is, the closer this variance comes to the stationary case. Hence, \( k \) can be chosen via a trade-off between the minimizing the additional overhead that the above computations represent, and minimizing the maximum probability of false alarms during the initial time steps.

6 Example
To illustrate the FlexDx framework, let us consider the simulated example system shown in Figure 1, where a DC-servo is connected to a flywheel through a rotational (damped) spring. The system dynamics can be described by:

\[
J_1 \ddot{\theta}_1(t) = k u(t) - \alpha_1 \dot{\theta}_1(t) - M_s(t) \\
M_s(t) = \alpha_2 (\dot{\theta}_1(t) - \dot{\theta}_2(t)) + \alpha_3 (\hat{\theta}_1(t) - \hat{\theta}_2(t))
\]

\[
J_2 \ddot{\theta}_2(t) = -\alpha_4 \dot{\theta}_2(t) + M_s(t)
\]

where \( u(t) \) is an input signal controlling the torque from the motor (with a scaling coefficient \( k = 1 \)), \( \theta_1(t) \) and \( \theta_2(t) \) are the angles of the motor axis and the flywheel, respectively, and \( M_s(t) \) is the torque of the spring. The moments of inertia in the motor is \( J_1 = 1 \) and for the flywheel \( J_2 = 0.5 \). The parameters \( \alpha_1 = 1 \) and \( \alpha_2 = 0.1 \) determine the viscous friction at the motor and flywheel respectively, while \( \alpha_3 = 0.05 \) is the spring constant and \( \alpha_4 = 0.1 \) the viscous damping coefficient of the spring.

As outputs, the motor axis angle and velocity, and the angle of the flywheel are measured. We will design the diagnosis system for six possible single faults \( f_1(t), \ldots , f_6(t) \); one for each equation. The augmented system model becomes

\[
J_1 \ddot{\hat{\theta}}_1(t) = k(u(t) + f_1(t)) - \alpha_1 \dot{\hat{\theta}}_1(t) - M_s(t) \\
M_s(t) = \alpha_2 (\dot{\hat{\theta}}_1(t) - \dot{\theta}_2(t)) + \alpha_3 (\hat{\theta}_1(t) - \hat{\theta}_2(t)) + f_2(t)
\]

\[
J_2 \ddot{\hat{\theta}}_2(t) = -\alpha_4 \dot{\hat{\theta}}_2(t) + M_s(t) + f_3(t)
\]

\[
y_1(t) = \theta_1(t) + f_4(t) + v_1(t) \\
y_2(t) = \dot{\theta}_1(t) + f_5(t) + v_2(t) \\
y_3(t) = \dot{\theta}_2(t) + f_6(t) + v_3(t)
\]

Here, \( v_i(t) \), for \( i = 1, 2, 3 \), are measurement noise terms.

Since the diagnosis framework will work on sampled data, the model is discretized before designing the tests, using a zero-order hold assumption. The noise is implemented as i.i.d. Gaussian noise with variance \( 10^{-3} \). By using the second type of tests described in Section 4.3 for the discretized system, a set of 13 tests were needed and their fault sensitivity
is shown in Table 1. These tests will in the following simulations be combined with the second test selection method described in Section 4.4.

### 6.1 Test Reconfiguration

To show how the diagnosis system is reconfigured during a fault transient, we will describe what happens when the fault occurs at time in a simulated scenario. The course of events is described in Table 2.

Each row in the table gives the most important properties of one iteration in the FlexDx procedure given in Section 2. In one such iteration, the set of active tests are executed on observations collected from time to time . The column minimal diagnoses shows a simplified representation of the minimal diagnoses during the corresponding phase. Each iteration ends when one or several of the active tests trigger an alarm, these are shown in bold type.

Let us take a closer look at the steps of the FlexDx procedure. Step 1 initiates the set of minimal diagnoses to , which is shown in row 1. The degree of redundancy of the behavioral model for NF is 3, and therefore 3 tests are needed to check if with consistent. Step 2 computes the first, in lexicographical ordering, minimum cardinality solution to (8), which is the test set given in row 1. Step 3 initiates the tests and triggers an alarm at time . From the fault sensitivity of given in Table 1, becomes a conflict which is the output of step 4. The new set of minimal diagnoses, computed in step 5, is shown in the second row. Returning to step 2, the degree of redundancy for each of the behavioral models corresponding to minimal diagnoses are 2, and therefore at least two tests are needed to check the consistency of each. The minimum cardinality test set computed in step 2 is . This set is shown in row 2. Tests 1 and 3 check the consistency of , 1 and 10 the consistency of , 3 and 13 the consistency of , and 10 and 13 the consistency of . In step 3, the last fault free time is estimated to by using the alarming residual . The initial states of the residuals used in the tests are estimated using observations sampled in a time interval ending at . Proceeding in this way, FlexDx finds in row 4 that is the only consistent single fault and then the multiple fault diagnoses are further refined.

### 6.2 Reduction of the Computational Burden

In a simulated scenario, the system is started in the fault-free mode. At , and at , is set to 0.1. The residuals computed by the diagnosis system are shown in Figure 2. It is noteworthy that the residuals have not been computed for all time-points. By comparing the number of residuals computed for a diagnosis system running all tests at all times with the number of residuals computed with the proposed system, a 78.3% reduction in the number of computed residuals is obtained for the simulated scenario. This number is in itself not an indication of expected computational gain in a typical application. For systems with low failure rate, more redundancy, or more complex system model the reduction will typically be much larger. The key point is that not all tests are run at all times, and during fault free operation, typically only a few tests are needed. The largest number of tests is performed during the fault transitions which lasts only a short period of time.

### 7 DyKnow

To implement an instance of the FlexDx framework, a number of issues have to be managed besides implementing the algorithms and integrating them to a system. When a potential fault is detected, FlexDx computes the last known fault free time and the new set of residual generators to be monitored starting at time . To implement this, three issues have to be solved. First, the FlexDx instance must be reconfigured to replace the set of residual generators and their monitors.

---

Second, the computation of the residuals must begin at time $t_f$ in the past. Third, at the same time as FlexDx is computing residuals and performing tests on the historic data, system observations will keep coming at their normal rate.

To manage these issues, FlexDx is implemented using DyKnow, a stream-based knowledge processing middleware framework for implementing applications processing asynchronous streams of information [7; 8].

DyKnow provides both a conceptual framework and an implementation infrastructure for integrating a wide variety of components and managing the information that needs to flow between them. It allows a system to incrementally process low-level sensor data and generate a cohesive view of the environment at increasing levels of abstraction. Due to the need for incremental refinement of information at different levels of abstraction, we model computations and processes within the knowledge processing framework as active and sustained knowledge processes. The complexity of such processes may vary greatly, ranging from simple adaptation of raw sensor data to controllers to diagnosis algorithms.

The system being diagnosed by FlexDx is assumed to be asynchronous. At the same time the diagnosis procedure is asynchronous, jumping back and forth in time trying to figure out which fault has occurred. This requires knowledge processes to be decoupled and asynchronous to a certain degree. In DyKnow, this is achieved by allowing a knowledge process to declare a set of stream generators, each of which can be subscribed to by an arbitrary number of processes. A subscription can be viewed as a continuous query, which creates a distinct asynchronous stream onto which new data is pushed as it is generated. Each stream is described by a declarative policy which defines both which generator it comes from and the constraints on the stream. These constraints can for example specify the maximum delay, how to approximate missing values or that the stream should contain samples added with a regular sample period. Each stream created by a stream generator can have different properties and a stream generator only has to process data if it produces any streams. The contents of a stream may be seen by the receiver as data, information or knowledge.

A stream-based system pushing information easily lends itself to “on-availability” processing, i.e. processing data as soon as it is available. This minimizes the processing delays, compared to a query-based system where polling introduces unnecessary delays in processing and the risk of missing potentially essential updates as well as wastes resources. This is a highly desired feature in a diagnostic system where faults should be detected as soon as possible.

For the purpose of modeling, DyKnow provides four distinct types of knowledge processes: Primitive processes, refinement processes, configuration processes and mediation processes. To introduce these processes and to describe how the three issues introduced by FlexDx are solved, we will use a concrete FlexDx instance as an example. An overview of the processes and streams is shown in Figure 3.

Primitive processes serve as an interface to the outside world, connecting to sensors, databases or other information sources that in themselves have no explicit support for stream-based knowledge processing. Such processes have no stream inputs but provide a non-empty set of stream generators. In general, they tend to be quite simple, mainly adapting data in a multitude of external representations to the stream-based framework. For example, in FlexDx the initial diagnosis and the stream of observations of the system being diagnosed are seen as a primitive processes System.

The second process type to be considered is the refinement process, which takes a set of streams as input and provides one or more stream generators producing refined, abstracted or otherwise processed values. In FlexDx there are four refinement processes, as seen in Figure 3:

- ResidualGenerator – Computes the residual for a particular test from system observations. The residual is initialized as described in Section 5.
- ResidualMonitor – Monitors a residual and checks whether it has triggered a test. This can either be a simple threshold check or a more elaborate test which checks properties of the residual over time, such as if it has been above or below the threshold for more than five consecutive samples. If a test has been triggered the process computes the last known fault free time; this is the output of the process.
- Diagnosis – Computes the new set of diagnoses each time a test has been triggered.
- TestSet – Computes the new set of residual generators to be monitored when the set of diagnoses changes.

The third type of process, the configuration process, takes a set of streams as input but produces no new streams. Instead, it enables dynamic reconfiguration by adding or removing streams and processes. In FlexDx a configuration process is required to handle the first issues, to be able to reconfigure the set of residuals and tests that are computed.

- CreateTests – Updates the set of residual generators and monitors as the set of tests changes. Each test consists of two refinement processes, one to compute the residual and one to monitor the test on the residual. In order to manage the second issue, that residuals are computed starting at the last known fault free time, the input to a residual is a stream which begins at this time-point. This is part of the policy the configuration process uses to set up the new residual generator process. Creating streams partially consisting of historic data is a DyKnow feature.
Finally, a mediation process generates streams by selecting or collecting information from other streams. Here, one or more of the inputs can be a stream of labels identifying other streams to which the mediation process may subscribe. This allows a different type of dynamic reconfiguration in the case where not all potential inputs to a process are known in advance or where one does not want to simultaneously subscribe to all potential inputs due to processing cost. FlexDx uses a mediation process to collect the detected conflicts:

- **ConflictSetMediator** – Subscribes to the output of each of the tests and aggregates these to a single stream. When tests are added or removed, the current set of subscriptions is updated accordingly. The output of this process is a stream of pairs, each pair containing the identifier of the test that was triggered and the last known fault free time for the corresponding residual.

FlexDx will continue to add new tests until there is exactly one consistent single fault or all tests have been added.

To give a concrete example of a run of the system, consider the example from Section 6 as described in Table 2. When the system is started, tests 1, 2, and 5 are created by CreateTests. These are computing the residuals and performing tests from time 0 to 102.6, when test 5 is triggered. Then the refinement process for test 5 computes the last known fault free time to 102.6. Using this information, Diagnosis computes the set of minimal diagnosis to \{1, 3, 5, 6\} and TestSet the new set of tests to \{1, 3, 10, 13\}. The old tests 1, 2, and 5 are removed and the new tests are added by CreateTests. All of the tests are computed from time 98.9 until time 102.7 when test 13 is triggered, which means that they are computed from historic data until time 102.6. In this manner the set of tests is updated one more time before concluding that \(f_1\) is the only consistent single fault. If there are no consistent single faults FlexDx will continue to add tests until all have been evaluated.

### 8 Summary

An implemented reconfigurable diagnosis framework FlexDx is proposed. It reduces the computational burden of performing multiple fault diagnosis by only running the tests that are currently needed. This involves a method for dynamically starting new tests. An important contribution is a method to select tests such that the computational burden is reduced while maintaining the isolation performance of the diagnostic system. Key components in the approach are test selection and test initialization. Specific algorithms for diagnosing linear dynamical systems have been developed to illustrate the diagnosis framework, but the framework itself is general.

Implementing a reconfigurable diagnosis framework such as FlexDx introduces a number of interesting issues. First, FlexDx must be reconfigured to compute the new set of tests each time the set changes. Second, these computations must begin at the last known fault free time, which will be in the past. Third, at the same time as FlexDx is performing tests on historic data, system observations will keep coming at their normal rate. To handle these issues FlexDx is implemented using DyKnow, a stream-based knowledge processing middleware framework.

In the given example, the proposed approach has shown a significant reduction of the computational burden for a relatively small dynamical system, and for larger systems the reduction is expected to be higher.

### References


Diagnosis of Coordination Faults: A Matrix-Based Formulation

Meir Kalech1 and Michael Lindner2 and Gal A. Kaminka2

1School of Engineering and Applied Sciences, Harvard University, MA USA kalech@eecs.harvard.edu
2Computer Science Department, Bar Ilan University, Israel lindnern@gmail.com, galk@cs.biu.ac.il

Abstract

One of the key requirements in many multi-agent teams is that agents must coordinate on specific aspects of their joint task. Unfortunately, this coordination may fail due to intermittent failures in sensor readings, communication failures, etc. A key challenge in model-based diagnosis of such coordination faults is to represent the model coordination between the agents in a way that allows efficient detection and diagnosis, based on observations of the agents involved. Previous mechanisms are useful only for small groups as they represent the coordination with binary constraints. This paper presents a model-based diagnosis (MBD) approach to coordination failure in which non-binary constraints are allowed. This model presents two advantages: (1) it appears quite frequently when modeling real problems, (2) it addresses large groups by gathering multiple coordinations in one constraint. To solve the diagnosis problem, we propose a matrix-based approach to represent the basic building blocks of the MBD formalization. This representation is both generic and efficient for large-scale teams.

1 Introduction

With increasing deployment of robotic and agent teams in complex, dynamic settings, there is an increasing need to be able to respond to failures that occur in multi-agent teams [Tambe, 1997; Kaminka and Tambe, 2000; Kalech and Kaminka, 2006]. One type of failure of particular interest in multi-agent systems is a coordination fault, where agents come to disagree on salient aspects of their joint task.

There is thus a particular need to be able to detect and diagnose the causes for coordination failures that may occur, in order to facilitate recovery and reestablishment of collaboration, e.g., by negotiations [Kraus et al., 1998]. This type of diagnosis is called social diagnosis, since it focuses on finding causes for failures to maintain social relationships, i.e., coordination failures.

In this paper we focus on a model-based diagnosis approach for coordination failures. Model-based diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987] relies on a model of the diagnosed system, which is utilized to simulate the behavior of the system given the operational context (typically, the system inputs). The resulting simulated behavior (typically, outputs) are compared to the actual behavior to detect discrepancies indicating failures. The model can then be used to pinpoint possible failing components within the system.

Previous work presents model-based diagnosis for coordination faults [Kalech and Kaminka, 2005; 2006], however, it models the coordination between every pair of agents as a set of binary constraints between the agents’ states. Thus, this representation does not scale well in the group size and in the number of states.

On the contrary, non-binary constraints appear quite frequently when modeling real problems [Bacchus and van Beek, 1998]. Such problems could be naturally defined by non-binary constraints between multiple agents. In addition, there are domains like RoboCup Rescue [Tambe et al., 2005] or ModSAF [Tambe, 1997], in which it may be more efficient to gather multiple coordinations (joint states) in one constraint rather than only one coordination per constraint. For instance, in RoboCup Soccer the players must coordinate the attack and the defense [Matsubara et al., 1998]. It is naturally defining the coordination between multiple attackers and multiple defenders and goalie with non-binary constraints. In addition, by a single constraint we can define the coordination between part of the actions of a defender with partial set of the attacker’s actions and the goalie’s actions.

In this paper we propose a model-based approach to address this kind of coordination setting. We model the desired behavior of a team, i.e., the allowed coordination among the agents. At runtime the agents are observed and by inferring their states and comparing to the allowed coordination model, we diagnose the faulty agents. To solve the diagnosis problem, we propose to use a matrix-based representation [Kalech et al., 2007] for the fundamental building blocks of the diagnosis problem. This representation has several benefits. First, it provides an easy and intuitive way to define the coordination between teammates. Second, since we do not represent the relations between teammates explicitly, but gather them compactly (joint coordination in the same matrix structure), this approach is scalable in the number of agents and states (unlike the approach proposed in [Kalech and Kaminka, 2005]). Finally, the use of a matrix-based representation, enables the use of the matrix operations and yields interesting information about the agents. To summa-
rize, the matrix representation enables an easy and efficient way to diagnose coordination failures.

2 Related Work

Kalech and Kaminka [Kalech and Kaminka, 2005] present a model-based diagnosis for general framework of coordination faults. In particular, they present consistency- and ablative-based approaches to this problem and propose distributed constraint satisfaction algorithms to solve the diagnosis problem [Kalech and Kaminka, 2006]. However, they model the coordination between the agents in pairs, meaning that their model grows exponentially in the group size and in the number of states.

Horling et al. [Horling et al., 1999] uses a fault-model of failures and diagnoses to detect and respond to multi-agent failures. In this model a set of pre-defined diagnoses are stored in acyclic graph nodes. When a fault is detected a suitable node is triggered and according to the fault characters the node activates other nodes along the graph. However, this work does not address the scale-up issues. In addition, the failure-model approach dictates that all possible failures be analyzed in advance.

Fröhlich et al. [Fröhlich et al., 1997] suggest dividing a spatially distributed system into regions, each under the responsibility of a diagnosing agent. If the fault depends on two regions the agents that are responsible for those regions cooperate in making the diagnosis. This method is inappropriate for dynamic team settings, where agents cannot pre-select their communication partners. Similarly, Roos et al. [Roos et al., 2004] analyze a model-based diagnosis method for spatially distributed knowledge. But, their method assumes that there are no conflicts between the knowledge of the different agents, i.e., that no coordination failure occurs.

Williams et al. [Williams et al., 2001; Kim et al., 2001] provide a model for cooperation of unmanned vehicles. They coordinate these vehicles by introducing a reactive model-based programming language (RMPL). This model is robust and can detect failures and recover. However, their model-based language addresses only smaller-scale systems.

In previous work [Kalech et al., 2007] we have proposed an approach to representing multi-agent coordination and observations, using matrix structures. This representation facilitates easy representation of coordination requirements, modularity, flexibility and reuse of existing systems. We have demonstrated how in principle, this representation can support detection of coordination faults. In this paper, we build on this work and utilize the matrix-based representation in model-based coordination diagnosis. We show that we can compactly represent joint states using matrix structures, and thus reduce (in part) the exponential complexity of the diagnosis to linear in the number of agents and states.

3 Fundamental Objects

We adopt a model-based diagnosis approach to diagnose the agents and the coordination failures. In model-based diagnosis of a single agent, the diagnoser uses a model of the agent to generate expectations which are compared to the observations, in order to form diagnoses [Reiter, 1987; de Kleer and Williams, 1987]. In model-based multi-agents diagnosis, the diagnoser models the coordination between the agents [Kalech and Kaminka, 2005]. The goal of the diagnosis is to diagnose the failures in the coordination by detecting deviation of the observation from the model’s predictions.

3.1 The Agent Model

The most fundamental entity is an agent. At any moment, an agent is found in a given state. This is a logical, internal representation of the agent status, or belief, at this very moment. Throughout the paper, we will refer to the following sets:

(i) Let \( A \) be a set of \( n \) agents, \( \{a_1, a_2, \ldots, a_n\} \).

(ii) Let \( S \) be set of \( m \) states, \( \{s_1, s_2, \ldots, s_m\} \).

For example, consider a management system for a shop consisting of the following six agents (hereinafter this example will be referred as "the shop"): Anny the manager, BENNY the cashier, two sellers (CANNY and DANNY), ERNY the storekeeper and a guard, FRENNY:

\[ A_{\text{shop}} = \{\text{ANNY}, \text{BENNY}, \text{CANNY}, \text{DANNY}, \text{ERNY}, \text{FRENNY}\} \]

Agents may be in one of eight possible states:

\[ S_{\text{shop}} = \{\text{BREAK}, \text{IDLE}, \text{NEGOTIATE}, \text{SELL}, \text{INNERTALK}, \text{WATCH}, \text{GUARD}, \text{EQUIP}\} \]

Having the two sets \( A \) and \( S \), we can define the environment for a team:

**Definition 1 (environment).** Let \( A \) be a set of agents, and let \( S \) be a set of states. The pair \( E = (A, S) \) is called the environment of \( A \) over \( S \).

Now that we have the definition of the environment, we can continue to define the relation between an agent and a state. In order to define the basic structures in terms of model-based diagnosis, we will use a first-order logic:

**Definition 2 (position).** A position function over an environment \((A, S)\) is a function that positions an agent in a particular state: \( \gamma : A \rightarrow S \). In terms of first order logic, we define the predicate \( \gamma(a_i, s_j) = \text{true} \equiv \gamma(a_i) = s_j \). We will use shorthand and denote \( \gamma(a_i, s_j) \) as \( s_j \).

As mentioned in the introduction, one of the novelties of this work is the possibility to gather joint coordination to one structure. To this end, we present a function to set multiple states for an agent. To this end, we will define superposition:

**Definition 3 (superposition).** A superposition function over some environment \( E = (A, S) \) is a function \( \Gamma : A \rightarrow \| S \| \setminus \emptyset \) i.e., it positions each agent in a set of possible states. Logically, \( \Gamma(a_i) = S_j \Longleftrightarrow \{s_j \in S_n \mid \gamma(a_i, s_j) = s_j\} \).

For example, let us refer back to the agents and states presented in the shop. \( \gamma(\text{ERNY}) = \text{GUARD} \) is a position \( \epsilon_{\text{GUARD}}(\text{ERNY}) \), while \( \Gamma(\text{ANNY}) = \{\text{INNERTALK, WATCH}\} \) is a superposition. In first order logic:

\[ \begin{align*}
\neg (\text{ANNY}) & \land (\text{INNERTALK}) \land (\text{BREAK}) \land \neg (\text{WALK}) \\
\neg (\text{ANNY}) & \land (\text{INNERTALK}) \land (\text{BREAK}) \land \neg (\text{WALK}) \\
\end{align*} \]

Figure 1 presents the full superposition function for the shop.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
\[ \Gamma(a) = \begin{cases} 
\{ \text{INNER TALK, WATCH} \} & a = \text{ANNY} \\
\{ \text{BREAK, SELL} \} & a = \text{BENNY} \\
\{ \text{BREAK, NEGOTIATE}, \text{SELL, EQUIP} \} & a \in \{ \text{CANNY, DANNY} \} \\
\{ \text{GUARD} \} & a = \text{ERNY} \\
\{ \text{BREAK, INNER TALK} \} & a = \text{FRENNY} 
\end{cases} \]

Figure 1: A superposition function.

### 3.2 A Model of Coordination

The multi-agent systems of interest to us are composed of several agents, which (by design) are to satisfy certain coordination constraints. We call this type of system a team, to distinguish it from general multi-agent systems in which it is possible that no coordination constraints exist.

The states of agents in a team are coordinated. We utilize a coordination primitive to define the coordination constraints. The coordination states a non-binary constraint between agents’ states, such that these states must be taken jointly, at the same time.

**Definition 4 (coordination(CRD)).** A coordination is a constraint between agents’ positions, requiring them to be true concurrently. Logically, we represent this constraint as follows: \( \text{CRD}(s_1, ..., s_n) \Rightarrow \langle s_1 \land \ldots \land s_n \rangle \)

For example, in the shop example above, an allowed coordination could be:

\[ \text{CRD}(\text{ANNY}_A, \text{BENNY}_A, \text{CANNY}_A, \text{DANNY}_A, \text{ERNY}_A, \text{FRENNY}_A) \]

Unlike [Kalech and Kaminka, 2005] that define a binary constraint to represent a coordination only for pair of agents, we define the coordination between multiple agents by a non-binary constraint. In addition, we allow joint coordination concurrently. That means that an agent can be found in one of multiple states while other agents can be found in multiple states. Fundamentally, we can represent the joint coordination as a conjunction statement of coordination constraints. However, it is more efficient to define them using superposition (Definition 3).

**Definition 5 (joint coordination).** A joint coordination is a constraint between agents’ super-position mandates that they must be true concurrently. We represent this constraint as follows: \( \text{CRD}(A, S) \Rightarrow \bigcup_{a \in A} (\Gamma(a)) = S' \subseteq S \). Logically:

\[ \text{CRD}(A, S) \Rightarrow \bigwedge_{a \in A} (\bigvee_{s_{ji} \in S'^i} s_{ji} \land (\bigwedge_{s_{j} \in S''} \neg s_{ji})) \]

The corresponding joint coordination for the superposition presented in Figure 1 is (only the true literals for each agent are shown):

\[ \text{CRD}(A, S) \equiv (\text{INNER TALK} \lor \text{WATCH}) \land \\
(\text{BREAK} \lor \text{SELL}) \land \\
(\text{BREAK} \lor \text{NEGOTIATE} \lor \text{SELL} \lor \text{EQUIP}) \land \\
(\text{GUARD}) \land \\
(\text{BREAK} \lor \text{INNER TALK}) \]

This representation allows defining multiple constraints between the agents in the same structure. For example, while ANNY selects state INNER TALK or WATCH, BENNY must select BREAK or SELL and so on for all the agents.

### 3.3 A Model of Actions

At any given moment, each agent is in a given state. As a result of its state, each agent takes some action, in order to fulfill its goal. An action is visible, i.e. others might observe it. A state is not necessarily related to one particular action. Rather, it is possible that one of a few given actions will be taken at service of the same state. In the opposite direction, the same action might be taken at service of a few different states. We will annotate the actions as a set \( B = \{ b_1, b_2, \ldots, b_t \} \).

For example, in the shop we define eight states logical positions of the agents and nine actions, which the agents might act upon. State SELL, for example, is when an agent is busy with closing the deal with a customer. Positioned at this state, the agent might act in one of the actions GET (getting the product off the shelf), CARRY (carrying it to the customer) or COUNTER (sitting near the counter). On the other hand, an agent might also CARRY or GET while positioned at state EQUIP, and not only when positioned in SELL.

When designing a multi-agent system, the designer defines which actions might be taken by an agent when positioned in each state. This is called the latitude of the agent.

**Definition 6 (latitude).** Let \( E = (A, S) \) be an environment, and \( B \) be a set of actions, the latitude of any agent \( a \in A \) is a function \( \lambda_a : S \rightarrow \| B \| \setminus \emptyset \).

This function maps, for any agent \( a \in A \) (rather than a certain agent as in definition 2), each state to a subset of actions which the agent is allowed to pick while being in this state. The straight-forward inverse function of \( \lambda_a \), the function \( \lambda_a^{-1} \), would map subsets of \( B \) to elements in \( S \). While this function is not interesting, we do define a kind of ‘inverse’ to the latitude function:

**Definition 7 (interpretation).** Let \( E = (A, S) \) be an environment, and \( B \) be a set of actions, the interpretation \( \nu_a : A \rightarrow \| S \| \setminus \emptyset \) is the function \( \lambda_a : B \rightarrow \| S \| \setminus \emptyset \). In terms of first order logic:

\[ (\lambda_a(b_j) = S' \subseteq S) \Rightarrow (\bigvee_{s_{ji} \in S'} s_{ji} \land (\bigwedge_{s_{j} \in S\setminus S'} \neg s_{ji})) \]

\( \lambda_a \) of a given action \( b_j \), is the set of all states that have \( b_j \) in their latitude. Given an action of any agent \( a' \), we interpret its action as one of a few given states, using this function. Figure 2 presents the latitude and interpretation functions for the shop example. For instance, an action Phone taken by any agent, say BENNY, implies that its states are BREAK or NEGOTIATE, meaning:

\[ \text{BENNY (BREAK \lor \text{NEGOTIATE}) \land} \\
\neg \text{BENNY (\lor \text{SELL} \lor \text{WATCH} \lor \text{GUARD} \lor \text{EQUIP})} \]

This is the first-order representation of the interpretation presented in Figure 2(b):
Diagnosis of Coordination Faults: A Matrix-Based Formulation

3.4 A Model of Observation

Knowing the exact state of each agent at every time requires that the agent reports its state any time it is changed. This is usually infeasible, since it involves massive communication resources. Our model-based diagnosis approach suggests looking at the action of each agent. Thus the last building block we define is the observation.

Definition 9 (agent-action). Let \( A = \{a_1, a_2, \ldots, a_n\} \) be a set of agents and \( B = \{b_1, b_2, \ldots, b_l\} \) a set of actions, an agent-action is a function \( \omega : A \to B \), that maps each agent to a particular action.

Definition 10 (observation (OBS)). A set of agent-actions:

\[ \text{OBS} = \{ (\omega(a_i) = b_k) \mid b_k \in B \land a_i \in A \} \]

In the shop example, the observation can be:

\[ \text{OBS} = \{ \omega(\text{Anny}) = \text{Stand} \}
\[ \omega(\text{Benny}) = \text{Stand} \]
\[ \omega(\text{Canny}) = \text{Phone} \]
\[ \omega(\text{Danny}) = \text{Get} \]
\[ \omega(\text{Erny}) = \text{Carry} \]
\[ \omega(\text{Frenny}) = \text{Walk} \] \]

4 Diagnosis of Coordination Faults

A fault in the coordination of a multi-agent system may be the result of a faulty agent(s). Given a \( MASD \) (Definition 8) it is possible to infer that a fault exists and to generate hypotheses as to the abnormal agents, by checking whether the observed actions of the agents satisfy the \( MASD \).

Let us formalize the coordination diagnosis in terms of model based diagnosis:

Definition 11 (Coordination Diagnosis Problem (CDP)). Given \( \{ A, MASD, OBS \} \) where \( A \) is a team of agents \( \{a_1 \ldots a_n\} \), \( MASD \) is a multi agent system description defined over \( A \) (Definition 8), and \( OBS \) is the set of the actions of the agents (Definition 10), then the coordination diagnosis problem (CDP) arises when

\[ MASD \cup \{ \neg AB(a_i) \mid a_i \in A \} \cup OBS \vdash \bot \]

Given a CDP, the goal of the coordination diagnosis process is to determine a minimal set of abnormal agents whose selection and subsequent setting of the \( AB(\cdot) \) clause would eliminate the inconsistency. To this end we define the consistency-based coordination diagnosis:

Definition 12 (consistency-based coordination diagnosis (CBCD)). A minimal set \( \Delta \subseteq A \) such that

\[ MASD \cup \{ \neg AB(a_i) \mid a_i \in A \} \cup OBS \vdash \bot \]

In our example, \( MASD \) is not consistent with the observation. A diagnosis for this coordination fault can be: \( \Delta = \{ \text{Erny}, \text{Frenny} \} \).

The goal now is to find \( \Delta \). Consistency-based minimal diagnosis is known as NP-hard problem [de Kleer and Williams, 1987]. In particular, Kalech and Kaminka have
proposed algorithm to find consistency-based coordination diagnosis [Kalech and Kaminka, 2005]. However, in their paper the coordination is represented by binary constraints between pair of agents’ states. On the other hand, in this paper we represent joint coordination setting (1) by a non-binary constraint between multi-agent, and (2) by joint coordination between multiple states of each agent, rather than single independent state. These two qualities enable an efficient representation of more realistic problems on the one hand, and on the other hand they simplify the representation so the diagnosis can be found even in linear time best case, in the number of agents and states. In the next section we propose a matrix-based representation presented in [Kalech et al., 2007], which uses as the basis for an algorithm for coordination diagnosis in linear time.

5 Matrix-Based Representation

We will represent the models of the coordination, the actions and the observation by matrices.

Let \( A = \{a_1, a_2, \ldots, a_n\} \) be a set of agents and \( S = \{s_1, s_2, \ldots, s_m\} \) be a set of states. We represent the joint coordination of the agents (Definition 5) by a Boolean matrix of order \( n \times m \).

**Definition 13 (coordination-matrix).** Let \( E \) be the environment \( \langle A, S \rangle \). A coordination-matrix \( C \) over \( E \) is a Boolean matrix of order \( n \times m \): \( \omega_{ij} = \begin{cases} 1 & s_i \in \Gamma(a_j) \\ 0 & \text{otherwise} \end{cases} \)

Given a set of states \( S = \{s_1, s_2, \ldots, s_m\} \) and a set of actions \( B = \{b_1, b_2, \ldots, b_l\} \), we can represent the interpretation of the actions to the states (Definition 7) by a Boolean matrix of order \( \ell \times m \).

**Definition 14 (interpretation-matrix).** Let \( S \) be a set of states and \( B \) a set of actions, an interpretation-matrix \( I \) from \( B \) to \( S \) is a Boolean matrix of order \( \ell \times m \): \( i_{ij} = \begin{cases} 1 & s_j \in A(b_i) \\ 0 & \text{otherwise} \end{cases} \)

Figure 4 presents the corresponding interpretation-matrix to the interpretation function presented in Figure 2(b). The rows represent the actions and the columns represent the states. For example, the second row says that once an agent is observed doing PHONE, then its state is one of \{BREAK, NEGOTIATE\}.

The last building block we define is the observation-matrix, which is parallel to the observation Definition (10) in the model-based diagnosis formulation.

**Definition 15 (observation-matrix).** Let \( A = \{a_1, a_2, \ldots, a_n\} \) be a set of agents and \( B = \{b_1, b_2, \ldots, b_l\} \) a set of actions, an observation-matrix \( \Theta \) stands for the observation matrix representation:

\[
\theta_{ij} = \begin{cases} 1 & \omega(a_i) = b_j \\ 0 & \text{otherwise} \end{cases}
\]

Figure 5 presents an example to an observation matrix. The rows represent the agents and the columns the actions. Pay attention that in every row there is exactly a single ‘1’ since every agent is observed in one action.

6 Diagnosis Procedure

A coordination fault occurs when the current agents’ positions (Definition 2) do not match the expected coordination given by the coordination-matrix (Definition 13). Thus, if we know the current positions of the agents, we can say for sure whether the system has a fault or not. The exact state of each agent is known only to the agent itself. However, its action is observable. By observing its current action, we can infer the state in which the agent is found. This could be done using the formula:

\[
\Omega = \Theta \cdot I
\]

Where, \( \Theta \) is the observation matrix and \( I \) is the interpretation matrix. \( \Omega \) is an \( n \times m \) Boolean matrix. Each element \( j \) in row \( i \) represents whether it is possible that agent \( a_i \) is now in state \( s_j \) (‘1’ entry) or not (‘0’ entry). Note that each element \( \omega_{i,j} \) is the sum of multiplying each element \( k \) in row \( i \) of \( \Theta \) by element \( k \) in column \( j \) of \( I \). This multiplication, of course, is ‘1’ if both of them are ‘1’... Since each row in \( \Theta \) has exactly one element which is ‘1’, the value of each element in \( \Omega \) will be at most ‘1’.

$\Omega \times \mathbb{R} = \Theta \cdot I = \begin{pmatrix}
\text{BREAK} & \text{IDLE} & \text{NEGOTIATE} & \text{SELL} & \text{INNER TALK} & \text{WATCH} & \text{GUARD} & \text{EQUIP} \\
\text{ANNY} & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
\text{BENNY} & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
\text{CANNY} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{DANNY} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\text{ERNY} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\text{FRENNY} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 
\end{pmatrix}$

Figure 6: The matrix given by the product between the observation-matrix and the interpretation-matrix.

$R = \Omega \cap C =$

$\begin{pmatrix}
\text{BREAK} & \text{IDLE} & \text{NEGOTIATE} & \text{SELL} & \text{INNER TALK} & \text{WATCH} & \text{GUARD} & \text{EQUIP} \\
\text{ANNY} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\text{BENNY} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{CANNY} & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{DANNY} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{ERNY} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{FRENNY} & 0 & 0 & 0 & 0 & 0 & 0 & 0 
\end{pmatrix}$

Figure 7: The matrix given by boolean ‘and’ operation between the coordination-matrix C and $\Omega$.

For example, Figure 6 presents the matrix given by the product between the observation-matrix (given in Figure 5) and the interpretation-matrix (given in Figure 4). Our observation may lead us to conclude that CANNY’s state is either BREAK or NEGOTIATE.

We can now explain the diagnosis algorithm. Failure is defined as a situation wherein none of an agent’s possible assigned state (according to $\Omega$) appears on the ‘allowed coordination’, designated as C (the coordination-matrix). In order to examine possible matches we will operate a logical ‘and’ between C and $\Omega$ in an element-by-element process, to get the results matrix, $R^{n \times m}$:

$$r_{ij} = c_{ij} \land \omega_{ij}$$  \hspace{1cm} (2)

$R$ represents all the agents-assigned combinations that satisfy C according to interpreted states by the observation. The combinations represented by $R$ are all those that agent $a_i$ is found in one of the states $s_j$ that match ‘1’ element in row $R_i$. Thus, if in each row $i$ in $R$ there is at least one ‘1’ element, it implies that at least one combination exists. In this case, we may assume that the agents will be found in one of those joint states. If, however, $R$ defines an all-zero row exists, then the assigned agents’ states are definitely forbidden. In this case, a failure alert is warranted, and the diagnosis is that the agents that are represented by these all-zero rows are abnormal. This operation takes only $O(nm)$ operations (counting the ‘1’s for $m$ elements on each of $R$’s $n$ rows).

Returning to the shop example, matrix $R$ in Figure 7 is the result of an element-by-element ‘and’ operation between C (Figure 3) and $\Omega$ (Figure 6). In this coordination-matrix, the two bottom lines, representing ERNY and FRENNY, are all-zero. No desired combination can explain their actions. A failure has been detected and the diagnosis is $\Delta = \{\text{ERNY, FRENNY}\}$.

In order to prove that the algorithm finds complete and sound diagnosis we will prove that all-zero row entails the abnormality of the agent represented by that row and vice versa. To prove this statement we should prove first two logical lemmas related to the consistency of the sets given by the superposition and the interpretation functions. To simplify the proof we define a set of states $S = \{s_1, s_2, \ldots, s_p\}$, and two subsets $S’, S’’ \subseteq S$ ($S’ \neq \emptyset, S’’ \neq \emptyset$), where $S'$ represents the set given by the superposition function and $S''$ represents the set given by the interpretation function. We define the following statements:

1. $ST_1 : (\bigvee_{s_j \in S'} s_j) \land (\bigwedge_{s_j \in S''} \neg s_j)$
2. $ST_2 : (\bigwedge_{s_j \in S'} s_j) \land (\bigwedge_{s_j \in S''} \neg s_j)$

Lemma 1. $S' \cap S'' = \emptyset \Rightarrow ST_1 \land ST_2 \perp \perp$

Proof: Without loss of generality, $ST_1 \Rightarrow \exists s_j \in S' = \text{true}$, but $S' \cap S'' = \emptyset \Rightarrow s_j \in S \setminus S''$. $ST_2 \Rightarrow s_j = \text{false}$. Consequently, $ST_1 \land ST_2 \perp \perp$.

Lemma 2. $S' \cap S'' \neq \emptyset \Rightarrow ST_1 \land ST_2 \not\perp \perp$

Proof: To prove consistency we need to show a truth assignment. Without loss of generality, assume $S' \cap S'' = s_1$, $ST_1 = \text{true} \land ST_2 = \text{true}$. Consequently, $ST_1 \land ST_2 \not\perp \perp$.

Theorem 1. Given a coordination-matrix representation: $\exists i, 1 \leq i \leq n : \bigwedge_{j=1}^{m} r_{ij} = 0 \Rightarrow AB(a_i)$

Proof:

1. $\exists i, 1 \leq i \leq n : \bigwedge_{j=1}^{m} r_{ij} = 0 \Rightarrow AB(a_i)$ (soundness):

   Without loss of generality, assume $\bigwedge_{j=1}^{m} r_{ij} = 0$ and prove that $AB(a_i)$.

   $\bigwedge_{j=1}^{m} r_{ij} = 0 \Rightarrow \forall j : c_{i,j} \land \omega_{i,j} = 0$ (equation 2).

   (a) $c_{i,j}$:

   i. $c_{i,j} = \Gamma(a_i) = S' \subseteq S (S' \neq \emptyset)$ (Definition 13).

   ii. $\Gamma(a_i) = S' \Rightarrow ST_1 = (\bigvee_{s_j \in S'} s_j) \land (\bigwedge_{s_j \in S \setminus S'} \neg s_j)$ (Definition 3).

   (b) $\omega_{i,j}$:

   i. $\omega_{i,j} = \Lambda(\omega(a_i)) = S'' (S'' \neq \emptyset)$ (equation 1, Definitions 14, 15).

   ii. $\Lambda(\omega(a_i)) = S'' \Rightarrow ST_2 = (\bigvee_{s_j \in S''} s_j) \land (\bigwedge_{s_j \in S \setminus S''} \neg s_j)$ (Definition 7).

   By (a) and (b): $\forall j : c_{i,j} \land \omega_{i,j} = 0 \Rightarrow S' \cap S'' = \emptyset$

   By Lemma 1: $ST_1 \land ST_2 \perp \perp$

   Consequently by Definition 8: $AB(a_i)$.

2. $\exists i, 1 \leq i \leq n : \bigwedge_{j=1}^{m} r_{ij} = 0 \Rightarrow \neg AB(a_i)$ (completeness):

   Without loss of generality, assume $r_{i,1} \neq 0$ and prove that $\neg AB(a_i)$.

   $r_{1,1} \neq 0 \Rightarrow c_{1,1} \land \omega_{1,1} = 1$ (equation 2).

   (a) $c_{1,1}$:

   i. $c_{1,1} = 1 \Rightarrow s_{1,1} \in \Gamma(a_1)$ (Definition 13).
Two sets of coordination $C_1 \cup C_2$, means that the set of allowed combinations in the system is the union of all the combinations defined by $C_1$ and all the combinations defined by $C_2$. This operator may be extended to expressions of the kind $C_1 \cup C_2 \cup \cdots \cup C_p$.

There may be cases in which the use of $\cup$ is more difficult for the designer to describe the system. Thus, we present the second basic operator, ‘and’, which is notated by $\wedge$. The expression $C_1 \wedge C_2$ represents all the combinations that are found in the intersection of those that are defined by $C_1$ and those defined by $C_2$. In fact, one might notice that any expression of the form $C_1 \wedge C_2$, may be reduced to an equivalent coordination-matrix, that represents exactly the same set of combinations. This is the coordination-matrix that is the result of a logical-and in an element-by-element fashion between $C_1$ and $C_2$.

We call this extended structure of combined coordination-matrices using operators a rule. An example for a complex rule is:

$$R = C_1 \cup (C_2 \cup C_3) \cap (C_4 \cup C_5) \cup C_6 \cup (C_7 \cup C_8 \cup C_9)$$

Back to the diagnosis problem, to find a diagnosis we should compare by ‘and’ing operator the product matrix of the interpretation-matrix and the observation-matrix ($\Omega$) against the coordination-matrix. Testing $\Omega$ against a rule $R = C_1 \cup C_2 \cup \cdots \cup C_p$, is simple. One must perform the all-zero row test presented earlier for each of the $p$ coordination-matrices. That is, for each $C_k$ in $R$, calculating the result matrix $R_k$ by logically ‘and’ing $\Omega$ with $C_k$ in an element-by-element fashion, and then check whether $R_k$ has all-zero row. Due to the nature of the operator ‘$\cup$’, it is enough to verify that at least one such $R_k$ has no all-zero row, in order to conclude that the agents are coordinated. For ‘and’ operator, on the other hand, (for instance $C_1 \wedge C_2$) the absence of the property of all-zero row must hold for both $C_1$ and $C_2$.

In fact we have shown [Kalech et al., 2007] an algorithm which reduces a rule to a collection of coordination-matrices that are all combined by an or operator. Thus we could detect failure by ‘anding’ each one of the coordination-matrices with $\Omega$, and check all-zero rows in the result matrices.

For the diagnosis purpose we should provide a set of abnormal agents. Based on the diagnosis definition we have presented here, an indication to a fault is once all the $p$ coordination-matrices produce all-zero rows in the corresponding $R$ matrices. Then each one of the matrices $R_k$ produces a diagnosis. For recovery purpose we prefer to explore minimal diagnoses. A minimal diagnosis is a diagnosis which no proper subset of it is a diagnosis. To this end, during the diagnosis process we prune all the diagnoses that are not minimal. In order to model complex rules in terms of model-based diagnosis we should define the $\sqcup$ and $\sqcap$ operators. Intuitively, since our model is defined in first order logic, we can define these operators using the regular logical operators $\lor$ and $\land$. A formal representation is beyond the scope of this paper.

7 Complex Coordination

One of the advantages of the matrix representation is the possibility to define complex coordinations [Kalech et al., 2007]. One coordination-matrix will usually not suffice for a full desired coordination definition. Thus, the coordination-matrix we introduced earlier (Definition 13), may only partially define the allowed combinations in a desired coordination. For instance, in the shop example, assume ERNY could replace FRENNY in GUARD duty, the coordination-matrix in Figure 3 does not deal with this new relation. Moreover, we cannot add another state to $C$, by just changing $c_{6,7}$ (FRENNY, GUARD) from ‘0’ to ‘1’. This would allow undesired combinations, such as ERNY and FRENNY guarding simultaneously. In this section we will briefly present the complex coordinations and then focus on the diagnosis aspects.

The most important operator used to join a few coordination-matrices is the ‘or’, notated as ‘$\sqcup$’. Defining
presented in this paper is more efficient and reflects the real world, by defining non-binary constraints between the agents and by enabling to gather multiple states in one constraint.

To solve the diagnosis problem we defined a matrix-based notation for the fundamental parts of the diagnosis representation, which serves as a general framework for coordination design and definition in multi agent systems. Using this representation, we showed an efficient fault detection and diagnosis algorithm in a space and time complexity that is linear by the number of agents and states.

In the future we plan to add partial observations capabilities which will find the minimum set of agents that may together provide the full information, or at least the best possible information. Combining this with explicit communication among agents may result a system that is cheap in resources, yet very reliable. In addition, at the moment our algorithm assumes that the coordination among the team members is defined at the beginning and must be consistent along the system lifetime. However, real-world multi-agent systems are dynamic, and the desired coordination may change, so we plan to extend our algorithm to dynamic coordination.

Another interesting field of future research is using probabilistic values for observations, rather than binaries. In this way, rather than defining the policy as ‘pessimistic’ or ‘optimistic’, we will be able to define the probability that a fault has occurred at a given moment.

Acknowledgements

Much of the work on this paper was done in the course of my postdoctoral fellowship at the School of Engineering and Applied Sciences at Harvard University. My deep appreciation goes out to Prof. Barbara Grosz for her support during this time, and to Rani Nelken for his help in the logic sections.

References


Local Consistency and Junction Tree for Diagnosis of Discrete-Event Systems

Priscilla Kan John*  
priscilla.kanjohn@anu.edu.au

Alban Grastien*  
alban.grastien@nicta.com.au

Keywords: Diagnosis, Discrete-event systems, Junction Tree, Local consistency

Abstract

We extend the decentralised/distributed approach of diagnosis of discrete-event systems modeled using automata. The goal is to avoid computing a global diagnosis, which is expensive, and to perform local diagnoses instead. To still ensure global consistency, we transform the topology of the system into a junction tree where each vertex represents a subsystem. Local consistency between the diagnoses of these subsystems ensures global consistency due to the tree structure. This technique will work best for systems whose natural structure is close to a tree structure, as the generated automata will be of reasonable size.

1 Introduction

Nowadays, many technical systems are highly automated, if not completely controlled by computers. As such systems increase in complexity, their supervision becomes more and more challenging such that there is a strong need to automate the task. New methods are required to meet this objective. We are here concerned with the model-based diagnosis of systems modeled as discrete-event systems (DES, [Cassandras and Lafortune, 1999]).

It is well-known that the diagnosis of discrete-event systems [Lamperti and Zanella, 2003] can be seen as the computation of all the trajectories on the model consistent with the observations. This can be done by unfolding the model according to the observations. The main challenge is then to cope with the complexity of the task as the representation of these trajectories is usually exponential in the number of components in the system.

To deal with systems of increasing size, several approaches have been investigated. A first approach trades time for space: the model of the system is compiled into a structure called the Sampath diagnoser [Sampath et al., 1995] to enable efficient on-line computation. However, this structure is double exponential in the number of components and cannot be built in most cases [Rintanen, 2007]. Use of symbolic tools has also been proposed, giving interesting results [Schumann et al., 2004; 2007; Grastien et al., 2007].

Another option is to consider local computations. Rather than computing the trajectories on the whole system, the trajectories are computed locally. The problem is then to make sure that the local sets of trajectories are consistent with each other. Unfortunately, local (pairwise) consistency does not ensure global consistency; worst, an algorithm that refines the local diagnoses pairwisely may not terminate. Methods were proposed to avoid global computation [Pencolé and Cordier, 2005; Cordier and Grastien, 2007; Su and Wonham, 2005; Fabre et al., 2005], but these methods do not scale up nicely.

The complexity of numerous algorithms in different domains drops when applied to trees. This is especially relevant to the case of ensuring global consistency as local consistency ensures global consistency on a tree structure. A popular solution to convert a graph into a tree is to make it into a junction tree [Huang and Darwiche, 1996], where the vertices are gathered in clusters. We thus transform the topological graph of the system into a junction tree where each cluster corresponds to a subsystem. The diagnosis is performed locally on each cluster, and local consistency is applied until a fixpoint is reached.

The paper is divided as follows: we first present basic notations on languages and diagnosis. In Section 3, we discuss the issues of distributed diagnosis, and the central notion of consistency. Our approach based on junction trees and local consistency is presented Section 4.

2 Preliminaries

In this section, we present basic notations on language and how it applies to the diagnosis of discrete-event systems.

2.1 Language Formalism

Let $\Sigma$ be any set. We denote $\Sigma^*$ the set of all finite sequences on $\Sigma$; an element $\sigma = e_1 \cdots e_n \in \Sigma^*$ is called a word over $\Sigma$; the empty word is denoted $\epsilon$. A language $L$ over $\Sigma$ is a subset of $\Sigma^*$. The projection on $\Sigma'$ of a word $\sigma$ over $\Sigma \supseteq \Sigma'$ denoted $P_{\Sigma \rightarrow \Sigma'}(\sigma)$ keeps all the elements of $\sigma$ in $\Sigma'$. Formally,

$$P_{\Sigma \rightarrow \Sigma'}(\sigma) = \begin{cases} 
\epsilon & \text{if } \sigma = \epsilon \\
P_{\Sigma \rightarrow \Sigma'}(\sigma') & \text{if } \sigma = e . \sigma' \text{ and } e \in \Sigma \setminus \Sigma' \\
é . P_{\Sigma \rightarrow \Sigma'}(\sigma') & \text{if } \sigma = e . \sigma' \text{ and } e \in \Sigma'
\end{cases}$$

---

*Priscilla Kan John and Alban Grastien are with National Information and Communications Technology Australia (NICTA) and The Research School of Information Sciences and Engineering, Australian National University, Canberra.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.  
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),  
September 22–24, 2008, Blue Mountains, NSW, Australia.
The projection on $\Sigma'$ of a language $L$ over $\Sigma$ is denoted $P_{\Sigma \rightarrow \Sigma'}(L)$ and defined by $\{P_{\Sigma \rightarrow \Sigma'}(\sigma) \mid \sigma \in L\}$. The inverse operation $P_{\Sigma \leftarrow \Sigma'}^{-1}(L)$ of the projection from $\Sigma$ to $\Sigma'$ generates all the finite words on $\Sigma$ whose projection on $\Sigma'$ is the parameter: $P_{\Sigma \leftarrow \Sigma'}^{-1}(L) = \{\sigma \in \Sigma' \mid P_{\Sigma \rightarrow \Sigma'}(\sigma) \in L\}$.

The synchronous product $\otimes$ between two languages $L_1$ over $\Sigma_1$ and $L_2$ over $\Sigma_2$ computes all the words over $\Sigma_1 \cup \Sigma_2$ whose projection on $\Sigma_i$ is $L_i$, i.e., $L_1 \otimes L_2 = \{\sigma \in (\Sigma_1 \cup \Sigma_2)^* \mid \forall i \in \{1,2\}, P_{\Sigma_i \leftarrow \Sigma_1 \cup \Sigma_2}(\sigma) \in L_i\}$.

The local consistency operation of language $L_1$ over $\Sigma_1$ on $L_2$ over $\Sigma_2$ denoted $cons_{\Sigma_1, \Sigma_2}(L_1, L_2)$ returns the minimum sublanguage of $L_2$ such that the synchronous product with $L_1$ is not modified: $cons_{\Sigma_1, \Sigma_2}(L_1, L_2) = \{\sigma \in L_2 \mid P_{\Sigma_2 \leftarrow \Sigma_1 \cup \Sigma_2}(\sigma) \in P_{\Sigma_1 \leftarrow \Sigma_1 \cap \Sigma_2}(L_1)\}$ or equivalently $cons_{\Sigma_1, \Sigma_2}(L_1, L_2) = L_2 \cap P_{\Sigma_2 \leftarrow \Sigma_1 \cap \Sigma_2}(P_{\Sigma_1 \leftarrow \Sigma_1 \cap \Sigma_2}(L_1))$.

### 2.2 Diagnosis of Discrete-Event Systems

We consider a system whose state can be described as the assignment of state variables over a discrete domain. We consider the evolution of the state variables to be discrete. The set of all — including unexpected — possible behaviours of this system is a language denoted $Mod$, which contains all the words over $\Sigma_1 \cup \Sigma_2$ whose projection on $\Sigma_i$ is $\Gamma_i$, i.e., $\Gamma_i \subseteq \Sigma_i \cup \Sigma_2$. The occurrence of an observable event generates an observation. While the system is running, it generates a flow of observations. The sequence of observable events that occur on the system is a word on $\Sigma_0$. However, it is not always possible to determine precisely the sequence of observable events from the sequence of observations received. We represent the sequence of observations by a language denoted $Obs$, where each word in $Obs$ is a sequence of observable events consistent with the observations received.

The diagnosis of the system is the problem of determining what possibly happened on the system given the observations on its behaviour. This can be simply computed by:

$$\Delta = Mod \otimes Obs.$$  \hspace{1cm} (1)

Languages can be represented by several tools. Regular languages are often represented by automata or Petri nets. The problem with these tools is that of state explosion. The size of these structures is exponential in the number of state variables, which makes them difficult to use in practice.

### 3 Consistency in a Distributed Model

Real-world systems are often distributed by nature, i.e., a set of interconnected components. The global behaviour of the system is complex, whereas each component has a simple behaviour. Recent approaches take advantage of this distributed nature to avoid computational blow up.

#### 3.1 Distributed Modeling

Modern technical systems are usually formed by combining simple components with simple behaviours leading to a device that exhibits complex behaviours. Rather than modeling the whole system, it is often preferable to model each component separately for many good reasons: fewer chances to make mistakes or forget behaviours, reusability, compactness.

Since the system is a set of components, each component $\gamma_i$ can be modeled separately: $Mod_i$ defined on alphabet $\Sigma_i$. Some formalisms consider that components share variables. Here, without loss of generality, we consider that components share events such that an event shared by several components must occur on each component at the same time. Other events may occur in a completely concurrent manner.

The system $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ composed of components $\gamma_1, \ldots, \gamma_n$ is modeled as a set of languages $dMod = \{Mod_1, \ldots, Mod_n\}$ over the alphabets $\Sigma_1, \ldots, \Sigma_n$. The global model of the system is implicitly defined by $Mod = Mod_1 \otimes \cdots \otimes Mod_n$, but never explicitly computed.

#### 3.2 Distributed Diagnosis and Global Consistency

The alphabet $\Sigma_i$ that represents the events of each component $\gamma_i$ is partitioned into observable events $\Sigma_{io}$ and unobservable events $\Sigma_{iu}$. Moreover, we consider that the global observations $Obs$ on the system can be distributed into $Obs_i$, defined on $\Sigma_i$, such that $Obs = Obs_1 \otimes \cdots \otimes Obs_n$.

A distribution $S = \{S_1, \ldots, S_m\} \in 2^\Gamma$ is a set of subsets of $\Gamma$ such that $S$ covers $\Gamma$: $S_1 \cup \cdots \cup S_m = \Gamma$. A distributed diagnosis is a mapping that associates with each subset $S_i$ a diagnosis $d\Delta(S_i)$ such that $d\Delta(S_1) \otimes \cdots \otimes d\Delta(S_m) = \Delta$. The literature usually considers that $S$ is a partition of $\Gamma$ [Pencolé and Cordier, 2005].

The local diagnoses can be simply computed by:

$$d\Delta(S_i) = \bigotimes_{\gamma_i \in S_i} (Mod_\gamma \otimes Obs_\gamma).$$  \hspace{1cm} (2)

This returns a distributed diagnosis that can be easily computed as long as any $S_i$ contains a small number of elements. However, the local diagnoses can be inconsistent with each other. Basically, some words of $d\Delta(S_i)$ should be removed because they disappear when $S_i$ is synchronised with other $S_j$ elements. Thus, we are interested in the globally consistent distributed diagnosis:

A distributed diagnosis $d\Delta$ is globally consistent if for all $i \in \{1, \ldots, m\}$, $d\Delta(S_i) = P_{\Sigma_1 \leftarrow S_{io}(\Delta)}$, where $S_{io} = \bigcup_{\gamma_i \in S_i} S_{io}^\gamma_k$.

The globally consistent distributed diagnosis is such that no word of any $d\Delta(S_i)$ can be removed. We want to compute this refined distributed diagnosis but the goal is to avoid the computation of $\Delta$.

#### 3.3 Local Consistency

The local consistency property requires that any pair of local diagnoses are consistent. Formally, a distributed diagnosis $d\Delta$ is locally consistent if $\forall\{S_1, S_2\} \subseteq S, P_{\Sigma_1 \leftarrow \Sigma_1 \cap \Sigma_2} (d\Delta(S_1)) = P_{\Sigma_2 \leftarrow \Sigma_1 \cap \Sigma_2} (d\Delta(S_2))$. It is possible to refine a distributed diagnosis using local consistency as presented in Algorithm 1. After the distribution is performed, and a local diagnosis is computed for each subsystem, the algorithm takes pairs of subsystems and performs a local consistency on these diagnoses. Basically, the idea is to remove the word of $d\Delta(S_i)$ that cannot be synchronised with any word of $d\Delta(S_j)$, and vice versa. The local consistencies can actually be performed in any order.
Algorithm 1 Distributed diagnosis algorithm based on local consistency

1: input $\Gamma$, $\{\text{Mod}_1, \ldots, \text{Mod}_n\}$, $\{\text{Obs}_1, \ldots, \text{Obs}_n\}$
2: $S = \{S_1, \ldots, S_m\} := \text{distribution}(\Gamma)$
3: for all $i \in \{1, \ldots, m\}$ do
4: $d\Delta(S_i) = \bigotimes_{k \in S_i} (\text{Mod}_k \otimes \text{Obs}_k)$
5: repeat
6: for all $\{S_i, S_j\} \subseteq S$ do
7: $d\Delta(S_i) = \text{cons}_{\Sigma_{S_i} \setminus \Sigma_{S_j}} (d\Delta(S_i), d\Delta(S_j))$
8: $d\Delta(S_i) = \text{cons}_{\Sigma_{S_i} \setminus \Sigma_{S_j}} (d\Delta(S_i), d\Delta(S_j))$
9: until $d\Delta$ is stable

However, as shown in [Su and Wonham, 2005], local consistency does not ensure global consistency. Moreover, because the languages may be infinite, no fix-point is reached in the worst case; the algorithm does not terminate. As noticed by the authors in [Su and Wonham, 2005], both problems disappear when the topology of the system forms a tree.

A topology of a distributed representation $S$ of the system is a graph $G = (V, E)$ where $V = S$ is the set of vertices and $E \subseteq V \times V$ is a symmetric and anti-reflexive set of edges such that $\forall (S, S') \in E, \forall e \in \Sigma_S \cap \Sigma_{S'}, \exists S_0, \ldots, S_{k+1}$ such that:
- $S_0 = S$ and $S_{k+1} = S'$
- $\forall i \in \{1, \ldots, k\}, e = \Sigma_{S_i}$, and
- $\forall i \in \{0, \ldots, k\}$, $\{S_i, S_{i+1}\} \in E$

Two subsystems that share an event are connected through an edge, or through a chain of edges where intermediate subsystems also share this event.

The graph $G$ is a tree if for any pair $S_i$ and $S_j$, there is exactly one path on the graph that contains no loop and leads from $S_i$ to $S_j$. Provided that the distribution of the system can be represented by a tree, the algorithm presented above terminates and is sound.

Because of space requirement, we only give a simplified proof of this last result. Similar proofs can be found in [Su and Wonham, 2005] with slightly different definition of the topology. In particular in [Su and Wonham, 2005], an edge connects two vertices whenever these two vertices share an event, while here we only required them to be connected through a chain of vertices that share this event.

Consider that the distribution generates a tree $G = (V, E)$. Consider that the local diagnosis $d\Delta(S_i)$ is computed for each subsystem $S_i \in V$ and that the local consistency procedure is applied until stability is reached.

Choose randomly some subsystem $S_i \in V$. We want to determine whether $P_{\Sigma_i \setminus \Sigma_{S_i}} (\Delta) = d\Delta(S_i)$ which states that the diagnosis is globally consistent. To do so, we set $S_i$ as the root of the tree $G$. Let $X \subseteq V$ be a subset of subsystems, we denote $\Sigma_X = \bigcup_{S \in X} \Sigma_S$ and $L_X = \bigotimes_{S \in X} d\Delta(S)$. We build $X$ incrementally from $X = \{S_1\}$ by adding $S_k \notin X$ such that $S_j \in X$ and $(S_k, S_j) \in E$; note that definition of the $\Sigma_i$ and $G$ since it is a tree, $\Sigma_X \cap \Sigma_k = \Sigma_j \cap \Sigma_k$.

Let $X' = X \cup \{S_j\}$. We prove by induction that for any $S_p \in X$, $P_{\Sigma_{X'} \setminus \Sigma_{S_p}} (L_X) = d\Delta(S_p)$. This is clearly the case for $X = \{S_i\}$.

4 Diagnosis by Junction Tree

4.1 Junction Tree

The concept of the junction tree is borrowed from the field of probabilistic inference where its structure is useful for working in complex domains [Huang and Darwiche, 1996]. Note that junction trees are also referred to as join trees in the literature [Schumann and Huang, 2008].

Definition 1 (Junction Tree) Let $G = (V, E)$ be a graph. A junction tree for $G$ is a pair $(T, C)$, where $T$ is a tree and $C$ is a function which maps each node $i$ in tree $T$ into a label $C_i$, called a cluster. The junction tree must satisfy the following properties:

1. $C_i \subseteq V$, i.e. each cluster is a set of vertices from $G$.
2. If two vertices are connected in $G$, they will appear together in some cluster $C_i$.
3. If a vertex appears in two clusters $C_i$ and $C_j$, it must also appear in every cluster $C_k$ on the path connecting vertices $i$ and $j$ in the junction tree. This is known as the running intersection property.

The separator of edge $i-j$ in a junction tree is defined as $C_i \cap C_j$. The width of a junction tree is the size of its largest cluster minus one.

One of the steps in obtaining a junction tree from a graph is to triangulate the graph, i.e., add extra links such that every cycle of length greater than three has a chord. There are different ways to triangulate a graph, yielding different sets of clusters. Moreover, each triangulated graph may have several different junction trees. It is therefore desirable to have optimal triangulations and optimal junction trees with respect to complexity. As discussed later, the complexity here depends on the size of the clusters: an optimal junction tree minimises the size of the largest cluster. However, the optimality problem for triangulation is NP-complete. Given a triangulated graph, we can obtain an optimal junction tree using an algorithm from [Jensen and Jensen, 1994] which is quadratic in the number of cliques.
The Figure 1 gives an example of three graphs and their junction trees. Note that the $i$th junction tree is also a junction tree for the $j$th graph if $i > j$ while it is not true if $i < j$. The best junction tree is the first, as its biggest cluster contains three elements against five for the last; furthermore, the first and second junction tree have the same largest cluster CDG, but the second largest cluster of the first junction tree is smaller than that of the second junction tree.

![Graph 1](image1)

**Graph 1**

![Junction Tree for Graph 1 (JT 1)](image2)

**Junction Tree for Graph 1 (JT 1)**

![Graph 2](image3)

**Graph 2**

![Junction Tree for Graphs 1, 2 (JT 2)](image4)

**Junction Tree for Graphs 1, 2 (JT 2)**

![Graph 3](image5)

**Graph 3**

![Junction Tree for Graphs 1, 2, 3 (JT 3)](image6)

**Junction Tree for Graphs 1, 2, 3 (JT 3)**

Figure 1: Three graphs and corresponding junction trees

The reasoning behind the use of junction trees in diagnosis is that it could help avoid the need to compute a global diagnosis. Using a junction tree representation of a system has two main advantages [Su and Wonham, 2005]:

1. A tree representation of a system implies that local consistency is equivalent to global consistency.
2. Non-termination issues with local consistency algorithms can be resolved.

### 4.2 Distribution Algorithm

The junction tree algorithm returns a topology as defined previously, provided it is followed by computation of the edges of the tree itself. Indeed, let $e \in \Sigma_{S_i} \cap \Sigma_{S_j}$ be an event that is shared by subsystems $S_i$ and $S_j$. We prove that any vertex $S$ in the path between $S_i$ and $S_j$ contains this event ($e \in \Sigma_S$). There are two (possibly identical) components $\gamma_1$ and $\gamma_2$ such that $\forall i \in \{1, 2\}, e \in \Sigma_i$ and $\gamma_i \in S_i$. Since component $\gamma_1$ and $\gamma_2$ share an event, they are connected in the original topology and because of the second property of junction trees, there is a cluster $S$ in the junction tree that contains both components ($\{\gamma_1, \gamma_2\} \subseteq S$). By the third property of the junction tree, all clusters between $S_i$ and $S$ contain component $\gamma_i$ and thus event $e$. $S$ can be between $S_i$ and $S_j$ or outside, but in both cases there is a path of clusters between $S_i$ and $S_j$ that share event $e$. Thus, the junction tree algorithm returns a tree-shaped distribution.

**Algorithm 2 Distribution using Junction Tree Algorithm**

1: input $\Gamma$, $\{\text{Mod}_1, \ldots, \text{Mod}_n\}$
2: $\mathcal{V} := \Gamma$
3: $\mathcal{E} := \{\{V_i, V_j\} \in \mathcal{V}^2 \mid i \neq j \& \Sigma_i \cap \Sigma_j \neq \emptyset\}$
4: $\mathcal{S} := \{\}$
5: while $\mathcal{V} \neq \emptyset$
6: pick a vertex $V \in \mathcal{V}$
7: $\mathcal{C} := \{\{\} \cup \{V\} \cup \{V, V'\} \mid \{V, V'\} \in \mathcal{E}\}$
8: $\mathcal{E} := \mathcal{E} \cup \{\{V_1, V_2\} \mid V_1 \in \mathcal{C}, V_2 \in \mathcal{C}\}$
9: $\mathcal{V} := \mathcal{V} - \{V\}$
10: $\mathcal{E} := \mathcal{E} - \{\{V_1, V_2\} \in \mathcal{E} \mid V_1 = V \lor V_2 = V\}$
11: if not ($\exists C' \in \mathcal{S} \mid C' \subseteq C'$) then
12: $\mathcal{S} := \mathcal{S} \cup \{C\}$
13: return $\mathcal{S}$

We perform distribution by rearranging the topology of the system into a junction tree, as described in Algorithm 2. We first obtain a graph of the original system, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each component $\gamma$ in the system is a vertex $V$ on the graph. The edges, $\mathcal{E}$, on the graph represent connected components. We use the junction tree algorithm [Huang and Darwiche, 1996] to obtain the clusters that make up $\mathcal{S}$. We pick a vertex $V \in \mathcal{V}$. A cluster $\mathcal{C}$ is obtained by taking the set formed by $V$ and its neighbours, i.e. the vertices on the graph that are connected to $V$ by an edge. We add edges so that all the vertices that make up a cluster are connected. $\mathcal{C}$ is added to $\mathcal{S}$ if it is not a subset of an element of $\mathcal{S}$. We update the original graph by removing $V$ and its associated edges from it. This procedure is repeated until no more vertices are left on the original graph. It is then trivial to calculate the separators that link the clusters into a junction tree.

As mentioned, building an optimal junction tree is NP-complete. However, we can use heuristics in the vertex selection phase of the algorithm (line 6) that would achieve polynomial-time while still producing a high quality tree [Huang and Darwiche, 1996]. One heuristic is to minimise the number of edges added to the graph [Kjørulf, 1990] (line 8 of the algorithm), which then achieves a low-polynomial complexity.

We mentioned in section 3.3 that local consistencies can be performed in any order. However, we can use a strategy, global propagation [Huang and Darwiche, 1996], that would
only require two ordered series of local consistency computations on the junction tree to achieve global consistency. We consider a message pass from a cluster $C_X$ to its neighbour $C_Y$ to be an operation that makes the components of $C_X$ locally consistent with those of $C_Y$. By performing these message passes in an ordered manner, we ensure that the consistency introduced by previous message passes is preserved.

4.3 Discussion

Using a junction tree is very interesting as the resulting subsystems tends to be of small size. However, this does not necessarily imply that the local diagnoses will actually be small as we show in the next example.

Consider a tree with $n$ nodes $N_1$ to $N_n$. Each node $N_i$ is associated with events $e_{i-1}$ and $e_i$. The topology of the tree is thus simply a line as node $N_i$ shares event $e_i$ with node $N_{i+1}$. The automaton of each node $N_i$ is represented in Figure 3. Since the initial state is the same as the final state, the number of occurrences of event $e_i$ is twice that of event $e_{i-1}$ for any $i$. Consider the node $N_1$ runs $k \in N$ loops. Then, event $e_0$ occurs $k$ times, event $e_1$ occurs $2 \times k$ times, etc. Event $e_{i}$ occurs $2^i \times k$ times. The globally consistent automaton representation the behaviour on node $N_i$ must represent the fact that event $e_{i-1}$ occurred $2^{i-1} \times k$ times and the event $e_i$ occurred $2^i \times k$ times for any natural number $k$ (and not for rational non natural numbers). This requires $2^{i-1} + 2^i$ states and transitions. In this example, the number of states after local consistency is exponential in the number of nodes.

The result basically comes from the fact that the events $e_i$ and $e_j$ in this example are not concurrent events but they occur in sequence. We expect that most systems actually exhibit concurrent behaviours. In this case, the size of the local diagnosis on a cluster is a direct function of the number of events attached with this cluster, and thus smaller cluster lead to a better efficiency.
diagnosis of discrete-event systems. If the distribution generates a tree-shaped topology, an algorithm based on local consistency can ensure global consistency of the diagnosis. We used the graph theory of junction trees to obtain good distributions. The complexity of the diagnosis is then often bounded by the tree width of the system topology which places an upper bound on the number of automata to synchronise together, though counter-examples exist.

We think there is still room for improvement. First, we proposed a static construction of the junction tree based only on the topology of the system. We want to investigate a more flexible technique where the junction tree is built after diagnoses and simple pruning operations are performed locally on components. The idea is that some connections in the system topology can be removed when no communication happened through these connections, leading to a graph with a smaller tree width. Moreover, we could then assign weight on each vertex of the graph. These technique should then improve the efficiency of diagnosis. More generally, we want to investigate more dynamic computations of junction trees: experiments have shown that the connections can often be removed after the distributed diagnosis is computed during the local consistency algorithm. For this reason, we want to start the diagnosis algorithm while the junction tree is being computed so as to dynamically change the construction of the junction tree. This is not trivial as the construction of the junction tree must satisfy some properties.

Regarding system design, an interesting exploration would be to interact with the system designer to propose alternative topology structures in the system in order to ensure a reasonable tree width of the system.

Finally, we considered that the observations emitted by different components were completely independent. However, it is often the case that a (partial) order exists between the observations. E.g. the alarm emitted by component 1 was surely emitted before the alarm from component 2. This generates some kind of connection between the two components and potentially interconnect all the components. We want to investigate this issue and determine when these connections can be removed, possibly with an approach based on time slicing [Cordier and Grastien, 2007].

6 Acknowledgements

This research was supported by NICTA in the framework of the SuperCom project. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References


Statistical Properties and Design Criterions for Fault Isolation in Noisy Systems

Mattias Krysander and Mattias Nyberg
Department of Electrical Engineering, Linköping University,
SE-581 83 Linköping, Sweden,
{matkr,matny}@isy.liu.se

Abstract

Fault diagnosis in the presence of noise and model errors is of fundamental importance. In the paper, the meaning of fault isolation performance is formalized by using the established notion of coverage and false coverage from the field of statistics. Then formal relations describing the relationship between fault isolation performance and the residual related design parameters are derived. For small faults, the measures coverage and false coverage are not applicable so therefore, a different performance criterion, called sub-coverage, is proposed. The performance of different AI-based fault isolation schemes is evaluated and it is notably shown that the well known principle of minimal cardinality diagnosis gives a bad performance. Finally, some general design guidelines that guarantee and maximize the fault isolation performance are proposed.

1 Introduction

The FDI (Fault Detection and Isolation) problem, as often described within the control community, is to detect and isolate any possible faults given sensor and actuator signals only. A typical solution, see Gertler [1998]; Patton et al. [2000]; Blanke et al. [2003], is to use a set of thresholded residuals together with a fault isolation scheme, which, based on the fact that the thresholded residuals respond differently to different faults, isolates the fault.

In a real application, there are typically model errors and noise. This fact limits our ability to construct a diagnosis system that perfectly detects and isolates the present fault. However, there is also design freedom available such as the threshold levels, the set of residuals to be included, and which isolation strategy to use. Thus, under the premises of noise and model errors, the design freedom should be utilized such that the ability of detecting and isolating faults is optimized.

The discussion above reveals first of all, that there is a need for an exact measure of FDI performance. Secondly, it is important to understand how this FDI performance changes when different design parameters are changed. In the literature, only a few studies have addressed these issues. In Nyberg [1999], FDI performance was studied in the framework of structured hypothesis tests. In Cordier et al. [2004] these issues were posed as open questions.

In several works, e.g. Nyberg and Krysander [2003]; Ploix et al. [2003]; Cordier et al. [2004], it has been recognized that fault isolation in FDI can be solved by using algorithms developed within the field of AI, see Kleer and Williams [1987]; Reiter [1987]. Advantages of these AI algorithms, compared to their counterpart from the control community, e.g. Gertler [1998], are that they can easily handle multiple faults and their computational efficiency. Because of these advantages we have in the present paper chosen to focus entirely on fault isolation algorithms from AI. However, the results can be easily generalized to cover fault isolation techniques from the control community such as structured residuals Gertler [1998].

In the paper, a first contribution is to formalize what we mean by FDI performance, especially for noisy and uncertain systems. For this we use the established notion of coverage and false coverage from the field of statistics. Then as a second contribution, we derive formal relations describing the relationship between FDI performance and the residual related design parameters. Further it is noted that a different performance criteria is needed for the smallest faults, and we therefore introduce a third performance measure called sub-coverage. We then discuss the intrinsic FDI performance of different AI-based fault isolation schemes. It is notable that the well known principle of minimal cardinality diagnosis gives a bad performance for the the smallest faults. Based on the performance measure and investigations, we develop some general design guidelines that, if followed, guarantee and maximize the fault isolation performance. Finally we illustrate the theory and the guidelines on a small application example.

2 Stochastic view on diagnosis

In many papers, both from the control community Gertler [1998]; Patton et al. [2000]; Blanke et al. [2003] and especially in AI Kleer et al. [1992]; Cordier et al. [2004], the systems to be diagnosed are assumed not to contain noise. This means that an observation in the model is either deterministic given the states, or completely unknown, depending on if a fault is present and also which fault that is present. The view taken here is that a system contains stochastic parts which implies that, given the states, observations have probability distributions rather than exact values. Based on this idea we will below give a basic stochastic framework for diagnosis.

Alban Graastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
2.1 The System

The system to be diagnosed consists of a number of components, and we assume here that the behavioral mode of a component is either non-faulty or faulty, abbreviated NF and F respectively. The behavioral mode of the complete system, called system behavioral mode or simply mode, can be described by a vector of length equal to the number of components, e.g., in a system with 5 components the system behavioral mode could be [NF, F, NF, NF, F].

Further, we assume that the system has a vector-valued trajectory \( z \) which is possible to observe. The vector \( z \) includes measured sensor values and actuated control values.

2.2 The Diagnosis System

We consider a diagnosis system to be a system that takes an observation as input and computes candidates, i.e. a set \( C \) of system behavioral modes, as output. The candidate set \( C \) is assumed to be a function of the observation and supposed to be the system behavioral modes that are likely explanations of the observation.

Formally we define observation as follows.

Definition 1 (Observation) An observation \( z_T \) of \( z \) is samples of \( z \) at times specified by the index set \( T \).

Here we assume \( T \) to be a finite set. Examples of \( T \) are \( T = \{0\} \) and \( T = \{0, 1, 3\} \).

Since we have a stochastic view on diagnosis, we consider \( z_T \) to be a random variable. For each system behavioral mode, we assume that \( z_T \) has exactly one given pdf (probability density function), denoted \( f_b(\cdot) \). Later in Section 6 we will relax this assumption. Since the candidate set \( C \) is a function of the observation \( z_T \), also \( C \) is a random variable which for each mode will have a unique pdf.

3 Statistical Performance Measures of Diagnosis Systems

Two performance measures of set estimators known from statistical decision making theory Casella and Berger [1990] will here be introduced as performance measures for diagnosis systems regarding their fault isolation capability. Note that in these performance measures, fault detection becomes a special case of fault isolation so we will refer only to fault isolation performance from now on.

3.1 Coverage Probability

Suppose that we want to diagnose a system that is operating in an unknown mode. It is almost never possible for a diagnosis system to exactly determine the present mode. A more realistic objective is that the candidate set \( C \) should at least with some high probability contain the present mode and the first performance measure formalizes this idea.

Definition 2 (Coverage Probability) Given a diagnosis system computing the candidate set \( C \), the coverage probability is a function of \( b \) given by

\[
P(b \in C | b) \tag{1}
\]

Practical Relevance of Coverage

Let NF denote the fault free system behavioral mode. False alarm can formally be described as the negation of coverage with respect to the mode NF. Thus the probability of false alarm becomes \( P(\text{NF} \notin C | \text{NF}) \). False alarms lead to expensive and unnecessary troubleshooting. Further, they degrade both the perceived product quality and the confidence in the diagnosis system. Therefore false alarms are in general not accepted in industrial applications.

Consider next the event \( b \notin C \) in the case that the present mode is \( b \) where \( b \neq \text{NF} \). If the user of the diagnosis result takes action based on the fact that \( b \) can not be the present mode, severe and expensive mistakes might be done. For example, if a repair technician excludes the possibility that \( b \) is the present mode, he will replace non-faulty parts and still not succeed with his repair mission.

From this discussion it is clear that lack of coverage is in general not acceptable in industrial applications.

3.2 False Coverage Probability

It is not sufficient to evaluate the isolation performance of a diagnosis system by using only its coverage probabilities. For example, a diagnosis system that always outputs that all system behavioral modes are candidates would have coverage probability 1 for all modes. Ideally we also want the candidate set \( C \) to exclude all modes that are not the present mode.

Definition 3 (False Coverage Probability) Given a diagnosis system computing the candidate set \( C \), the false coverage probability is a function of \( b \) and \( b' \) given by

\[
P(b' \in C | b), \quad \text{where } b' \neq b \tag{2}
\]

Note that, in contrast to coverage probability which is a function defined on the set of all modes, the false coverage probability is a function defined on the set of all non-equal pair of modes.

Practical Relevance of False Coverage

False coverage means that \( b' \in C \) even though another mode \( b \) is the present one. This is of course not a desired situation since it implies that the user of the diagnosis result has to undertake unnecessary safety or repair actions or to convey further analysis to exclude the mode \( b' \). However we consider it not as serious as lack of coverage.

4 Diagnosis Systems using AI-Based Fault Isolation

As said in the introduction, we consider diagnosis systems consisting of a set of diagnostic tests together with a fault isolation scheme using techniques from the field of AI. Further, we consider diagnostic tests in the view of hypothesis testing in accordance with Nyberg [1999]. It should be noted that this view is compatible with traditional fault isolation techniques from both FDI and AI, see Cordier et al. [2004].

The main idea is the following. Each diagnostic test \( \delta_k \) is a hypothesis test with a null hypothesis \( H_k^0 \) and a rejection region \( R_k \). The diagnostic test takes an observation \( z_T \) as input and generates a binary decision as output as follows. If \( z_T \in R_k \), then \( H_k^0 \) is rejected, otherwise \( H_k^0 \) is not rejected. The null hypothesis \( H_k^0 \) is here represented as a set of system
behavioral modes. When the null hypothesis is rejected, the
conclusion from the diagnostic test is that none of the modes
in $H_0^k$ is the one that has generated the observation $x_T$, i.e.
the present mode must be in the complement set $H_0^C$. Using
AI terminology, a rejected null hypothesis $H_0^k$ is a so called
conflict.

In the isolation scheme, the conclusions from the individual
diagnostic tests are merged. In its simplest form, the isolation
scheme is a simple intersection of the conclusions from the
tests, i.e.

$$C = \bigcap_{k \in \Omega_b} H_0^C$$

(3)

This principle has been used in both FDI and AI Nyberg
[1999]; Cordier et al. [2004] even though more efficient rep-resentations and computations have been utilized.

For an example, let $F_2$ denote the system behavioral mode
with a fault in component 2 only, let $F_12$ denote the sys-
tem behavioral mode with faults in components 1 and 2 only,
etc. Then consider the following table which we call decision
structure:

<table>
<thead>
<tr>
<th>NF</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F12</th>
<th>F23</th>
<th>F13</th>
<th>F123</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_1$</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$\delta_3$</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

A 0 in row $i$ and column $j$ means that the mode of column $j$
is a member of the null hypothesis of the test corresponding to
row $i$, i.e. $H_0^i$. Assume that $F_2$ is the present mode and that
the null hypotheses of the tests $\delta_1$ and $\delta_3$ have been rejected.
Then, according to (3),

$$C = H_0^C \cap H_0^C =$$

$$\{F_1, F_2, F_12, F_23, F_13, F_123\} \cap$$

$$\{F_2, F_3, F_12, F_23, F_13, F_123\} =$$

(5)

A problem with the fault isolation scheme (3), and as seen
even in this small example, is that the candidate set $C$ will in
general be very large and include many other modes in addition
to the present one. This problem is well known and has in the
field of AI been solved by, in a second step\footnote{Note
that computationally, this filtering (i.e. focusing) does not
necessarily need to be implemented as a second step.}, filtering out
less likely modes from $C$. This is often called focusing and is
based on the idea of a preference relation $\succeq$ defined on the
set of system behavioral modes.

For example, in (5), if single faults are preferred over mul-
tiple faults, the result is a focused set of candidates $C_F =
\{F_2\}$, which is actually the perfect result since $F_2$ was the
mode assumed to be present. Formally, the set $C_F$ can be de-

$$C_F = \{b \in C | \neg \exists b' \in C : b' \succ b\}$$

(6)

The preference relation $\succeq_b$ can be defined using different
principles of which the concepts of minimal diagnoses Kleer
and Williams [1987]; Reiter [1987]; Hamscher et al. [1992] and
minimal cardinality diagnoses Tuhrim et al. [1991] are the
two most common. In Section 7, these preference relations
and also the case without focusing, i.e. (3), will be compared
with respect to the fault isolation performance measures pre-

5 Bounds for the Performance Measures

In this section we will present bounds for the performance
measures presented in Section 3. The idea of these bounds
is to estimate the performance measures (1) and (2) by using
only the performance of the individual diagnostic tests. The
performance of each diagnostic test is specified in terms of the
probability $P(\text{reject } H_0^k \mid b)$ which, in the field of statistics, is
called power function Casella and L.Berger [1990]. For conve-
nience we will use the shorter writing $P(\text{rej}_k \mid b)$.

The rationale behind bounds of this type is that the design
freedom in designing diagnosis systems of the type described
in Section 4 lies in the selection and construction of the diag-
nostic tests. Thus, it is critical to know the relationship be-

$$P(\text{reject } H_0^k \mid b)$$

(7)

In the decision structure, $\Omega_b$ is the rows with 0 in column $b$.
For example, in (4), $\Omega_{F_2} = \{1, 2\}$.

Basic probability theory gives the general relations $P(A) +
P(B) - 1 \leq P(A \cup B) \leq \min(P(A), P(B))$ and $\max(P(A),
P(B)) \leq P(A \cap B) \leq P(A) + P(B)$ for two arbitrary events
$A$ and $B$. Using these relations we can derive the bounds given
in the following theorem.

Theorem 1 Let $B$ be the set of modes that are more preferred
than mode $b$, i.e. $B = \{b' \succ b\}$. If $\Omega_b \subseteq \Omega_b$ for some
$b \in B$, then

$$P(b \in C_F \mid b') = 0$$

(8)

for all $b'$. Otherwise, for mode $b'$ it holds that

$$1 - [B] - \sum_{k \in \Omega_b} P(\text{rej}_k \mid b') + \sum_{b \in B} \max_{j \in \Omega_b} P(\text{rej}_j \mid b')$$

$$\leq \min \left(1 - \max_{k \in \Omega_b} P(\text{rej}_k \mid b'), \min_{b \in B} \sum_{j \in \Omega_b} P(\text{rej}_j \mid b') \right)$$

(9)

The proof of Theorem 1 as well as all other results in the
paper can be found in Nyberg and Krysander [2007]. Note that
no assumption about the correlation between the response of
different tests has been made in the theorem above.

From Theorem 1 a number of bounds can be derived both
for coverage probability and false coverage probability. For
example if a bound for coverage probability in the case of no
focusing is needed, let $b = b$ and $B = \emptyset$.

Later in the paper we will use the following simplified upper
bound for false coverage probability.

Corollary 1 (False Coverage Probability) It holds that

$$P(b \in C_F \mid b') \leq 1 - \max_{k \in \Omega_b} P(\text{rej}_k \mid b')$$

(10)
Next, by using the assumption
\[ P(\text{rej}_b | b) = 0, \text{ for all } b \in H_b^b \] (11)
a simplified lower bound for coverage probability can be derived. Note that (11) implies that we assume that the false alarm probability is zero.

**Corollary 2 (Coverage Probability)** Assume that (11) holds and let \( \mathbb{B} \) be defined as in Theorem 1. If \( \Omega_b \subseteq \Omega_b \) for some \( b \in B \), then
\[ P(b \in C_F | b) = 0 \] (12)
for all \( b \). Otherwise, it holds that
\[ 1 - |B| + \max_{b \in \mathbb{B}} P(\text{rej}_b | b) \leq P(b \in C_F | b) \] (13)

### 6 Relaxing the Assumption of Unique Distributions

In Section 2.1 we assumed that \( z_T \), and consequently \( C \) and \( C_F \), have exactly one given pdf for each mode \( b \). This assumption is quite restrictive since it requires that the behavior of a fault is relatively well known. Thus it is desirable to relax this assumption. We do this here by assuming that for a specific mode \( b \), the random variable \( z_T \) has a pdf in a set \( \Phi_b \).

The next issue is the performance measures presented in Section 3. For example, the coverage probability \( P(b \in C_F | b) \) is no longer well defined since the fact that \( b \) does not give a single distribution for \( z_T \) and consequently not for \( C_F \). Our solution to this problem is to instead consider a coverage probability conditioned on one specific distribution in the set \( \Phi_b \). Thus we write
\[ P(b \in C_F | z_T \sim f_b(z_T)) \] (14)
For convenience we will mostly write \( P(b \in C_F | f_b(z_T)) \) instead of (14). When using the coverage probability measure, and only the set \( \Phi_b \) is specified, we do not get a single coverage probability for a specific mode \( b \) but instead a set, possibly infinite, of coverage probabilities. Thus, the next question is how to use such a performance measure.

First, note that a mode \( b \) may contain both small and large faults. For example consider the mode bias of a sensor. There are both small biases, close to zero and large ones. Because we consider stochastic noisy systems, it is not realistic to require good performance for both small and large faults. For example to require that the diagnosis system detects and uniquely isolates a very small bias is not realistic, but it may be realistic to require both good detection and isolation for large biases. Thus, the required performance of a diagnosis system need to be formulated differently for small and large faults respectively.

Formally, we start by partitioning the set \( \Phi_b \) into two subsets \( \Phi_b^{\text{sig}} \) and \( \Phi_b^{\text{insig}} \), representing significant faults and insignificant faults respectively. We will below use different performance requirements for these two sets. The idea of this partitioning is that \( \Phi_b^{\text{sig}} \) contains the pdf’s of those faults that are critical to detect and isolate. The set \( \Phi_b^{\text{insig}} \) is then the pdf’s of the faults that neither need to be detected or isolated.

Note that the partitioning into significant faults and insignificant faults may be the result of an FMEA. Typically small faults are classified as insignificant and large as significant but in principle this must not be true. For instance, it can very well be the case that the set of significant faults \( \Phi_b^{\text{sig}} \) contains some very small faults, even though this probably makes it harder to design a diagnosis system that fulfills requirements associated with the significant faults.

### 6.1 Performance Measures for Significant Faults

For each pdf belonging to \( \Phi_b^{\text{insig}} \), we use the following measures corresponding to coverage and false coverage probability respectively:
\[ P(b \in C_F | f_b(z_T)) \] (15)
\[ P(b \in C_F | f_b(z_T)) \] (16)
Still, the number of performance measures will typically be infinite. A solution to handle this is given later, together with the application example, in Section 9.

### 6.2 Performance Measure for Non-significant Faults

For the distributions belonging to \( \Phi_b^{\text{insig}} \), we use another performance measure. To explain this, assume that the present fault in the system is insignificant, i.e. associated with a distribution in \( \Phi_b^{\text{insig}} \). Then if \( \text{NF} \) is not present in \( C_F \), i.e. a clear indication of that the system is faulty, then a reasonable requirement is that at least some mode in \( C_F \) should indicate that there is a fault in some of the components that are indeed faulty. If this would not be the case, \( C_F \) would indicate a fault but only in a part of the system not related to the present fault, which would for example completely mislead a mechanic trying to repair the system.

To achieve this, we will, for modes in \( \Phi_b^{\text{insig}} \), not aim at strict coverage. Instead we aim only for something that we will call sub-coverage. Further we do not care about false coverage at all which means that if \( b \) is the present mode, it is acceptable to also have other modes \( b' \) included in \( C_F \).

The idea of sub-coverage is that we consider it fully acceptable to say that a component is non-faulty even though it is faulty. For example, if \( b = [\text{NF}, F, \text{NF}, F] \) is the present mode and \( z_T \) has a distribution belonging to \( \Phi_b^{\text{insig}} \), it is acceptable if \( [\text{NF}, F, \text{NF}, F, \text{NF}] \) \( \notin C_F \) as long as \( [\text{NF}, \text{NF}, \text{NF}, F], [\text{NF}, F, \text{NF}, F], [\text{NF}, F, \text{NF}, \text{NF} \text{NF} \text{NF}] \) belong to \( C_F \).

To formalize this, use \( \psi \) to denote the behavioral mode of the \( i \)th component which means that \( b \) can be written as \( b = [\psi_1, \psi_2, \ldots, \psi_n] \). Then let \( \leq_O \) be a relation\(^2\), defined on the set of system behavioral modes, such that \( b' \leq_O b \), where \( b' = [\psi'_1, \psi'_2, \ldots, \psi'_n] \), if and only if \( \forall i \in \{1, 2, \ldots, n\} : \psi'_i = \text{NF} \lor \psi'_i = \psi_i \). By using this relation we replace the performance measure of coverage probability (15) with a measure that we call sub-coverage probability:
\[ P(\exists b \in C_F : b \leq_O b | f_b(z_T)) \] (17)

### 6.3 Bounds for Sub-Coverage

The aim now is to derive a useful bound for the probability of sub-coverage. We do this for the special case when the preference relation \( \geq_p \) is such that \( b' \geq_p b \) implies \( b' \leq_O b \).

---

\(^2\)If system behavioral modes are represented by their sets of faulty components the relation \( \leq_O \) is equivalent to the subset relation.
Theorem 2 If the preference relation $\succeq_p$ is such that $b' \geq_p b$ implies $b' \leq_O b$, then for any $f_b(z_T) \in \Phi_b$, it holds that
\[
P(\exists b \in C : b \leq_O b | f_b(z_T)) \geq P(b \in C | f_b(z_T)) \tag{18}
\]
PROOF If $b \in C$ then there is a mode $b'$, where $b' \geq_p b$, and $b' \in C_F$. Since it holds that $b' \geq_p b$ implies $b' \leq_O b$, it follows that
\[
\exists b \in C_F : b \leq_O b \tag{19}
\]
Thus, we have proven that $b \in C$ implies (19). This fact means that (18) holds trivially. □

As seen this theorem shows that if we aim for coverage in $C$ we get also sub-coverage.

7 Comparison of Focusing Principles

In this section we will compare the diagnosis system performance when using minimal and minimal cardinality diagnosis as focusing strategies and also the case without focusing. We use the performance measures defined in the previous section, i.e. coverage, false coverage, and sub-coverage. For sake of simplicity, we assume that (11) holds.

7.1 No Focusing

First, consider the strategy to not use focusing, i.e. $C_F = C$. Since we assume that (11) holds, the bound (13) with $B = \emptyset$ gives directly that $P(b \in C_F | f_b(z_T)) = 1$. Since $C_F \subseteq C$ also $P(b \in C | f_b(z_T)) = 1$, which implies, according to Theorem 2, that also $P(\exists b \in C_F : b \leq_O b | f_b(z_T)) = 1$. Thus both coverage and sub-coverage are guaranteed.

In general, false coverage can not be avoided. A typical example is if $[F, NF, NF]$ is the present mode. Then, assuming we have coverage, it holds that $[F, NF, NF] \in C$ but also that $[F, F, NF] \notin C$ since it is typically not possible to construct a diagnostic test which responds to the mode $[F, NF, NF]$ but not to $[F, F, NF]$. Thus the system says that mode $[F, NF, NF]$ is preferred and thus, $[F, F, NF] \notin C_F$. The underlying idea of this focusing principle is that if a diagnosis system says that mode $[F, NF, NF]$ is consistent with observations, there is no reason to believe that the a-priori much less probable mode $[F, F, NF]$ is the present mode.

All discussions in Section 7.2 regarding coverage and false coverage performance are valid for the case minimal diagnosis focusing. In addition we can note that, as a direct consequence of Theorem 2, the probability of sub-coverage is always greater than the coverage probability when minimal diagnoses focusing is used.

7.2 Focusing

We saw in the previous section that the strategy of no focusing gives perfect performance with respect to coverage and sub-coverage, but very bad false coverage performance. The bad false coverage performance is the reason why focusing is used and we will in this section quantify how focusing improves the false coverage performance but also how the coverage performance is reduced if no special care is taken. We will later, in Section 7.3 and 7.4, see also that the sub-coverage performance may be severely affected depending on the actual focusing strategy chosen.

First consider the coverage probability. If $NF$ is more preferred than any other mode, which should hold in any sensible focusing strategy, coverage in the case the present mode is $NF$ is guaranteed from the bound (13) since the set $B$ will be empty and we assume that (11) holds. For other modes, we do not get coverage automatically. When mode $b$ is present, we need the tests to respond in a way such that all modes $b >_p b$ are eliminated from $C_F$. A sufficient condition to achieve coverage with high probability is obtained from the bound (13). This relation says that for each $b >_p b$ it is sufficient to have one test that responds to $b$ but not to $b$ with high probability. Then the sum will be close to $|B|$ which implies that the bound becomes close to 1. Thus, the selection and design of a set of tests with this property for all significant faults is critical to obtain high coverage probability.

As said above, the only reason to use focusing is to lower the probability of false coverage. Given a mode $b$, consider the modes $b$ for which it holds that $b <_p b$ or $b >_p b$. For these modes it holds that $b \in C_F$ implies $b \notin C_F$. Therefore we have $P(b \notin C_F | f_b(z_T)) \geq P(b \in C_F | f_b(z_T))$. Thus, if we aim for high probability of coverage of $b$, which is of primary importance, we get also low false coverage probability of the pair $(b, b)$.

Next, if $b \not\succeq_p b$ and $b \not\succeq_p b$, low false coverage probability can be guaranteed via the upper bound in (9) or the simplified bound (10). If the simplified bound is used, it tells us that a sufficient condition to get low false coverage probability is to, for each mode $b$ where $b \not\succeq_p b$ and $b \not\succeq_p b$, have one test with $b \in H^b_\emptyset$ and which responds with high probability when $b$ is present.

7.3 Minimal Diagnoses

Now consider the case of focusing by means of the principle of minimal diagnoses Kleer and Williams [1987]. This principle says that $\succeq_p = \leq_O$. That means for example that if $[F, NF, NF] \in C$ and $[F, F, NF] \in C$, the mode $[F, NF, NF]$ is preferred and thus, $[F, F, NF] \notin C_F$. All discussions in Section 7.2 regarding coverage and false coverage performance are valid for the case minimal diagnosis focusing. In addition we can note that, as a direct consequence of Theorem 2, the probability of sub-coverage is always greater than the coverage probability when minimal diagnoses focusing is used.

7.4 Minimal Cardinality Diagnoses

Next consider the focusing strategy minimal cardinality. This principle says that $b_1 \geq b_2$ if the number of faulty components in $b_1$ is less or equal to the number of faulty components in $b_2$. For example, $[F, NF, NF] >_p [NF, F, F]$. As in the case of minimal diagnosis focusing, all discussions in Section 7.2 regarding coverage and false coverage performance are valid for the case of minimal cardinality diagnosis focusing. However, there is an important difference regarding sub-coverage, something that is revealed by the following example. Assume that we have a diagnosis system with the following decision structure and that each test $\delta_k$ is designed to respond to the mode of a column if the row contains $X$ in the column.

<table>
<thead>
<tr>
<th></th>
<th>NF</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F12</th>
<th>F23</th>
<th>F13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_1$</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$\delta_3$</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Assume the mode $F_{23}$ is present with an insignificant fault and because the fault is small, only tests $\delta_1$ and $\delta_2$ respond. This implies that $C = \{F_1, F_{12}, F_{23}, F_{13}\}$. Minimal cardinality focusing gives $C_F = \{F_1\}$. It is obvious that sub-coverage is not obtained. Note that in the case of minimal diagnosis focusing, sub-coverage is obtained (even coverage) since $C_F = \{F_1, F_{23}\}$.

The important conclusion of this study is that if an insignificant fault is present, we have no control of whether tests respond or not, and thus we can not guarantee any level of sub-coverage probability when using minimal cardinality focusing.

## 8 Guidelines for Design of Diagnosis Systems

In Section 6 we have presented three fault-isolation performance-measures: coverage probability, false coverage probability, and sub-coverage probability. In this section we aim at giving some general design guidelines such that desired performances with respect to these three measures are obtained or maximized. First however we give some general presumptions as a starting point.

In Section 3.1 it was argued that lack of coverage can not be accepted in industrial applications. Therefore, but also to make our analysis tractable, we decide to aim for coverage probability one, i.e. $P(b \in C_F|f_b(z_T)) = 1$ for significant faults.

In Section 3.2 it was argued that false coverage is not as serious as lack of coverage. Therefore, and because we would often get an unsolvable problem if we would require false coverage with probability zero, we will not aim at $P(b \in C_F|f_b(z_T)) = 0$ when $b \neq b$ and the fault $b$ is significant.

Instead we aim at $P(b \in C_F|f_b(z_T)) \leq \epsilon$ where $\epsilon$ may be fixed or dependent on the pair $(b,b)$.

We assume that the diagnosis system design starts with a default set of diagnostic tests where each test $\delta_k$ has a residual generator $r_k$ and a set $H_k^b$. This situation is common for example if the diagnosis system design starts with a search for residual generators via structural analysis Krysander [2006].

The design freedom then consists of: (i) selecting the rejection region, i.e. the threshold and possibly some residual filtering, of each test $\delta_k$, (ii) from the default set select tests $\delta_k$ to be included in the diagnosis system, and (iii) to select the focusing strategy.

### 8.1 Selection of Rejection Region

A necessary requirement for coverage is that $P(b \in C|f_b(z_T)) = 1$ and from Theorem 1, it can be shown that a necessary and sufficient condition to achieve this is that the rejection region, for each diagnostic test $\delta_k$, fulfills

$$P(\text{rej}_k|f_b(z_T)) = 0 \text{ for all } b \in H_k^b \quad (21)$$

This rule is, as seen in Section 7.2, however not sufficient to obtain coverage in the case when focusing is used. When $b \geq p$, coverage can only be guaranteed if we also have at least one test that responds to $b$ but not to $b$. Further, from (10) it is clear that also to obtain low false coverage probability, it is important to have tests that responds as much as possible to modes $b \notin H_k^b$. These facts means that we must follow the constraint (21) but in addition, it is in general advantageous to maximize the probability $P(\text{rej}_k|f_b(z_T))$. This leads us to our first design guideline:

**G1.** For each diagnostic test $\delta_k$, select the maximal rejection region such that $P(\text{rej}_k|f_b(z_T)) = 0$ for all modes $b \in H_k^b$ and all distributions $f_b(\zeta_T) \in \Phi_b^{\text{insig}} \cup \Phi_b^{\text{sig}}$.

### 8.2 Selection of Diagnostic Tests to Include

Following design guideline G1 is necessary to obtain coverage but as seen in Section 7.2 not sufficient if focusing is used. As was stated above, a sufficient condition is to, for each pair of modes such that $b \geq p$, have at least one test that responds to $b$ with probability one but not to $b$. From Section 7.2 it has already been concluded that if coverage of a $b$ is secured, we only have to consider false coverage of modes $b$ where $b \not\sim p$ and $b \not\sim p$. This leads us to our next design guideline:

**G2.** For each pair of modes $(b,b)$, make sure that for all distributions $f_b \in \Phi_b^{\text{insig}} \cup \Phi_b^{\text{sig}}$ and $f_b \in \Phi_b^{\text{sig}}$ there is, included in the diagnosis system, at least one test $\delta_k$ such that $b \in H_k^b$, $P(\text{rej}_k|f_b(z_T)) = 0$, and

a) $P(\text{rej}_k|f_b(z_T)) = 1$ if $b \geq p$, $b$

b) $P(\text{rej}_k|f_b(z_T)) \geq 1 - \epsilon$ if $b \not\sim p$ and $b \not\sim p$.

### 8.3 Selection of Focusing Strategy

Note that a consequence of the discussion in Section 7.1 is that fulfillment of guideline G2 is in general not possible if we don’t use a focusing strategy. This implies that, of the three choices of no focusing, minimal diagnoses, and minimal cardinality diagnoses, we have to use minimal diagnoses or minimal cardinality diagnoses.

We have seen in Section 7 that the choice of focusing method affects the ability to obtain sub-coverage. Of the two choices left, i.e. minimal diagnoses and minimal cardinality diagnoses, minimal diagnosis is the best choice since it guarantees high sub-coverage probability when we have high coverage probability. This is our final design guideline:

**G3.** Use the focusing strategy minimal diagnoses.

### 8.4 Summarizing Theorem

We end this section by summarizing the discussion in a theorem.

**Theorem 3** If guidelines G1, G2, and G3 are followed, we obtain a diagnosis system where:

a) $P(b \in C|f_b(z_T)) = 1$ for all $f_b(z_T) \in \Phi_b^{\text{sig}}$ and for all $b$, i.e. coverage is guaranteed for all significant faults,

b) $P(\exists b \leq b : b \in C_F | f_b(z_T)) = 1$ for all $f_b(z_T) \in \Phi_b^{\text{insig}}$ and for all $b$, i.e. sub-coverage is guaranteed for all insignificant faults.

c) $P(\forall b \in C_F | f_b(z_T)) \leq \epsilon$ for all $f_b \in \Phi_b^{\text{insig}} \cup \Phi_b^{\text{sig}}$ and $f_b \in \Phi_b^{\text{sig}}$ and for all pairs $(b,b)$, i.e. false coverage probability less than $\epsilon$ is guaranteed.

Further, no other choice of rejection region for each test gives strictly better performance in all measures of coverage, sub-coverage, or false coverage probability.

### 9 Example

Consider a system with a pump $P$ and two sensors $S_1$ and $S_2$. The angular velocity $x$ of the pump is measured by sensor $S_1$. The angular velocity determines the output pressure which is
measured by sensor $S_2$. The measurement signals are denoted $y_1$ and $y_2$ respectively. All three components are assumed to be either in a non-faulty $NF$ or faulty mode $F$. The system behavioral modes are denoted by their faulty components, e.g. $S_1$ means the mode where only the sensor $S_1$ is faulty.

Next, we assume that the following model is available:

$$P = NF \rightarrow u_a = u \quad (22a)$$

$$\dot{x} = f(x) + u_a \quad (22b)$$

$$S_1 = NF \rightarrow y_1 = x \quad (22c)$$

$$S_2 = NF \rightarrow y_2 = g(x) \quad (22d)$$

$$S_3 = F \rightarrow y_2 = g(x) + c \quad (22e)$$

where $c$ is an unknown constant. Even though not written out explicitly we assume that all equations also are affected by noise terms with unspecified pdf’s. Note that, and as will be shown below, it is for our purpose not important to know these unspecified pdf’s explicitly.

According to our framework, the set of pdf’s $\Phi_b$, for each mode $b$, is assumed to be partitioned into two sets $\Phi_{b}^{aig}$ and $\Phi_{b}^{naig}$. However, in this example, these sets are not specified explicitly. Instead we pick out, from each set $\Phi_{b}^{aig}$, a pdf $f_{b}^{aig}(\tau)$ that represents a benchmark fault. Then the benchmark fault is defined explicitly and we assume that the pdf $f_{b}^{aig}(\tau)$ is representative for the whole set $\Phi_{b}^{aig}$ in the sense that for each $f_{b}(\tau) \in \Phi_{b}^{aig}$ it holds that $P(\text{rej}_{k}|f_{b}(\tau)) \geq P(\text{rej}_{k}|f_{b}^{aig}(\tau))$ for all $k$.

It is assumed that only modes $P$, $S_1$, $S_2$, and $S_1S_2$ are important to detect and isolate and thus, only these are considered to have significant faults and consequently also benchmark faults. The benchmark fault for mode $P$ is defined by replacing equation (22a) by $u_a = u + \Delta u_{\text{min}}$, and the benchmark fault for mode $S_1$ is defined by replacing equation (22c) by $y_1 = x + \alpha_{\text{min}}$. Further the benchmark fault for mode $S_2$ is defined by $c = c_{\text{min}}$. Finally, the benchmark fault for mode $S_1S_2$ is the combination of the benchmark faults for $S_1$ and $S_2$.

Next, structural analysis, see Krysander [2006], is used to find the equation sets that can be used to derive residual generators and their corresponding null hypotheses. The result is that 7 sets are found and the decision structure for potential tests $\delta_k$, to be constructed from these equation sets found, is the following.

<table>
<thead>
<tr>
<th>equation set</th>
<th>$NF$</th>
<th>$P$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_1S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_1$</td>
<td>(22a), (22b), (22c)</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>(22a), (22b), (22d)</td>
<td>$X$</td>
<td>$0$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$\delta_3$</td>
<td>(22c), (22d)</td>
<td>$0$</td>
<td>$0$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$\delta_4$</td>
<td>(22a), (22b), (22c), (22d)</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$\delta_5$</td>
<td>(22a), (22b), (22e)</td>
<td>$X$</td>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\delta_6$</td>
<td>(22c), (22e)</td>
<td>$0$</td>
<td>$0$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$\delta_7$</td>
<td>(22a), (22b), (22c), (22e)</td>
<td>$X$</td>
<td>$0$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
</tbody>
</table>

(23)

In the decision structure above only modes which have significant faults are shown. All other multiple-fault modes have $X$’s only in their columns.

### 9.1 Diagnosis System Design

Now we have all the elements needed to start the design of the diagnosis system. By following guideline G3 we will use the focusing strategy minimal diagnoses. By using guideline G2 we will now describe how to, from the list of potential tests (23), select a subset of tests $\Delta$ to be included in the diagnosis system.

Given the focusing strategy and the significant faults considered, it follows that there is one requirement in guideline G2a for each pair in $R_q = \{(NF,P), (NF,S_1), (NF,S_2), (NF,S_1S_2), (S_1,S_1S_2), (S_2,S_1S_2)\}$ and in guideline G2b, one for each pair in $R_{nc} = \{(S_1,S_1S_2), (S_1,P), (S_2,P), (S_1,S_2), (P,S_1), (P,S_2), (P,S_1S_2), (S_1,S_2,P)\}$. To illustrate how to fulfill these requirements, consider the pair $(S_2,S_1S_2) \in R_a$. To fulfill guideline G2 for $(S_2,S_1S_2)$ we need a test where $S_2 \notin H^f_k$. Potential tests fulfilling this are tests $\delta_k$ indexed $\{1,5,6,7\}$. Note that, since we intend to follow guideline G1, it will hold that $P(\text{rej}_{k}|f_{S_2}(\tau)) = 0$ for any test $\delta_k$, $k \in \{1,5,6,7\}$, if included in the diagnosis system. If we choose to include $\delta_k$, a consequence of fulfilling G1 is also that $P(\text{rej}_{k}|f_{S_1S_2}(\tau)) = 0$. This implies that, since we are looking for tests that fulfill G2a for the pair $(S_2,S_1S_2)$, there are only the potential tests $\{1,6,7\}$ left.

### 9.1.1 Focusing Strategy

Thus to fulfill guideline G2a for $\{1,6,7\}$, there are only the potential tests in $\pi_1 = \{1,6,7\}$ to be included in the diagnosis system.

For all other pairs in $R_a \cup R_{nc}$, sets $\pi_k$ of potential tests are obtained in the same way. A necessary requirement for a diagnosis system with tests $\Delta$ to fulfill G2, is that the set $\Delta$ has a non-empty intersection with all sets $\pi_k$.

By applying a minimal hitting set algorithm Kleer and Williams [1987], we get that the minimal test sets are $\{1,2,3,5\}, \{2,3,5,6\}$, and $\{2,3,5,7\}$. Hence a set of tests $\Delta$ included in a diagnosis system fulfilling G2 must necessarily be a superset of some of these minimal test sets. This is however not sufficient since both G2a and G2b specify requirements on $P(\text{rej}_{k}|f_{S_2}(\tau))$ for all $f_{S_2} \in \Phi_{b}^{aig}$.

Assume that we decide to investigate if the minimal test set $\{1,2,3,5\}$ fulfills the requirement on $P(\text{rej}_{k}|f_{S_2}(\tau))$ for all pairs in $R_a \cup R_{nc}$. For this set, all requirements on $P(\text{rej}_{k}|f_{S_2}(\tau))$ specified by G2a and G2b correspond to non-zero entries in the following table.

<table>
<thead>
<tr>
<th>$NF$</th>
<th>$P$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_1S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_1$</td>
<td>$p_1$</td>
<td>$p_4$</td>
<td>$0$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\delta_2$</td>
<td>$p_2$</td>
<td>$0$</td>
<td>$p_6$</td>
<td>$1$</td>
</tr>
<tr>
<td>$\delta_3$</td>
<td>$0$</td>
<td>$0$</td>
<td>$p_5$</td>
<td>$p_7$</td>
</tr>
<tr>
<td>$\delta_4$</td>
<td>$0$</td>
<td>$p_3$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Then from guidelines G2a and G2b we can derive the requirements that $\max(p_1,p_2,p_3) = 1$, $\max(p_4,p_5) = 1$, $\max(p_6,p_7) = 1$, and $p_i > 1 - \epsilon$ for all $i = 3, \ldots, 8$. The constant $\epsilon$ is the guaranteed false coverage probability that in this example is chosen as $\epsilon = 0.1$.

The next step is to construct residual generators for the selected equation sets and investigate if the requirements in (24) are achievable by filtering and thresholding of these residuals. Observer based residual generators are derived for $k = \{1,2,5\}$ and a static residual generator is derived using equation set 3 in (23). Then the pdf’s $f_{S_2}(\tau)$ corresponding to the benchmark faults are estimated using data from the real process. These estimated pdf’s are then used for selecting, by means of thresholding and filtering, the rejection region in accordance with G1.
Assume that there are thresholds for the residuals such that the following performance $P(\text{rej}_k|f^b_*(zT))$ for the benchmark faults has been confirmed:

\[
\begin{array}{cccc}
\text{NF} & P & S_1 & S_2 & S_1S_2 \\
0 & 1 & 1 & 0 & 1 \\
2 & 0 & 0.8 & 0 & 1 \\
3 & 0 & 0 & 0.95 & 0.97 \\
5 & 0 & 0.9 & 0 & 0 \\
\end{array}
\]

By using this matrix, the bounds for $P(b \in C|b^b)$ in Theorem 1, where $b^b$ corresponds to the rows and $b$ to the columns, are:

\[
\begin{array}{cccc}
\text{NF} & P & S_1 & S_2 & S_1S_2 \\
0 & 1 & 0 & 0 & 0 \\
1 & 0 & [0.05] & 0 & [0.01] \\
2 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 \\
\end{array}
\]

The interpretation of the first row is that, when the present mode is NF then $C_F = \{\text{NF}\}$ with probability 1. In row 3, we can see that when $S_1$ is the present mode then $S_1 \in C_F$ but $P$ will also be included in $C_F$ with a probability less than 0.05. No other modes will be included in $C_F$. All diagonal elements are 1, i.e. complete coverage of all significant faults have been obtained. All non-diagonal elements are less or equal to 0.1 and this means that the false coverage probability is less than 10%. In fact, the false coverage probability is better than the guaranteed 10% for all modes except for $P$.

10 Conclusions

The first contribution of the paper is the formalization of “fault isolation performance” in noisy and uncertain systems. For this we have used the established notion of coverage and false coverage from the field of statistics. Further it has been noted that a different performance criteria is needed for insignificant faults, and we have therefore introduced the third performance measure sub-coverage. We have also derived formal relations describing the relationship between fault isolation performance and the null-hypotheses and rejection regions of the tests. Further, the intrinsic fault isolation performance of different AI-based fault isolation schemes has been evaluated and it has notably been concluded that the well known principle of minimal cardinality diagnosis gives a bad performance for the case of small faults. Finally, based on the performance measure and investigations, we have developed some general design guidelines that, if followed, guarantee and maximize the fault isolation performance.

References


Pervasive Diagnosis: 
Integration of Active Diagnosis into Production Plans

Lukas Kuhn, Bob Price, Johan de Kleer, Minh Do, Rong Zhou

Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
{lkuhn,bprice,dekleer,minhdo,rzhou}@parc.com
Contact: 650-812-4802 FAX: 650-812-4334

Abstract
In model-based control, a planner uses a system description to create a plan that achieves production goals. The same model can be used by model-based diagnosis to indirectly infer the condition of components in a system from partially informative sensors. Existing work has demonstrated that diagnosis can be used to adapt the control of a system to changes in its components, however diagnosis must either make inferences from passive observations of production plans, or production must be halted to take specific diagnostic actions. In this paper, we observe that the declarative nature of model-based control allows the planner to achieve production goals in multiple ways. We show that this flexibility can be exploited by a novel paradigm we call pervasive diagnosis which produces diagnostic production plans that simultaneously achieve production goals while generating additional information about component conditions. We derive an efficient heuristic search for these diagnostic production plans and show through experiments on a model of an industrial digital printing press that the theoretical increase in information can be realized on practical real-time systems and used to obtain higher long-run productivity than a decoupled combination of planning and diagnosis.

1 Introduction
Artificial intelligence has long been animated by the vision of creating fully autonomous systems that not only act on the larger world, but also maintain and optimize themselves. The increased autonomy, reliability and flexibility of such systems are important in domains ranging from space craft, to manufacturing processes, to computer networks. Autonomy can be seen as the combination of two processes: diagnosis of the current condition of components in a system from weakly informative sensor readings and model-based control of system operation optimized for the current condition of the system. In an aerospace domain, flight dynamics models can be used to diagnose faults in flight control surfaces from noisy observations of flight trajectories. A model-based flight controller could then compensate for the faults by using alternative control surfaces or engine thrust to achieve the pilot’s goals. Diagnosis and model-based control are typically combined in one of two ways: 1) alternation of an explicit diagnosis mode with a model-based control mode or 2) parallel execution of a passive diagnosis process with a model-based control mode. The alternation of an explicit diagnosis mode with a model-based control mode typically results in long periods in which regular operation must be halted. This is particularly true when diagnosing rare intermittent faults. The combination of a passive diagnosis process with model-based control is often unsuccessful as regular operation may not sufficiently exercise the underlying system to isolate an underlying fault.

In this paper we introduce a new paradigm, pervasive diagnosis, in which certain parameters of model-based control are actively manipulated to maximize diagnostic information. Active diagnosis and model-based control can therefore occur simultaneously leading to higher overall throughput and reliability than a naive combination of diagnosis and regular operation. In this paper, we show that pervasive diagnosis can be efficiently implemented by combining model-based probabilistic inference together with our own unique decomposition of the information gain associated with executing a plan and an efficient heuristic search that exploits this decomposition. Our formulation turns out to be an efficient general solution to the problem of creating plans that produce maximal information gain from a system. In the next section we examine how Pervasive Diagnosis compares to existing work. In subsequent sections we describe pervasive diagnosis in detail, and explain a specific instantiation of pervasive diagnosis. The implementation of this instantiation is then demonstrated on a commercial scale printing system.

2 Related Work
The general mechanisms for inferring underlying causes of observations have a long history in artificial intelligence and engineering including logic based frameworks [15], continuous non-linear systems [11], probabilistic models for mobile robotics [17], xerographic systems [18], and hybrid logical probabilistic diagnosis [12]. In active diagnosis, specific inputs or control actions are chosen to maximize diagnostic information obtained from a system. The optimal sequence of tests has been calculated for static circuit domains [3]. This can be ex-
tended to sequential circuits with persistent state and
dynamics through time-frame expansion methods sec-
tion 8.2.2 of [2]. Finding explicit diagnosis tests can be
done using a SAT formulation [1]. The use of explicit
diagnosis jobs has been suggested for model-based con-
trol systems [7].
The combination of passive diagnosis to obtain informa-
tion and model-based control conditioned on the infor-
mation has appeared in many domains including auto-
matic compensation for faulty flight controls [14], choos-
ing safe plans for planetary rovers [4], maintaining wire-
less sensor networks [13] and automotive engine control
[10].
We are not aware of any other systems which explicitly
seek to increase the diagnostic information returned by
plans intended to achieve operational goals.

3 Pervasive Diagnosis

Pervasive diagnosis is a new paradigm in which pro-
duction is actively manipulated to maximize diagnos-
tic information. Active diagnosis and production can
therefore occur simultaneously leading to higher long
run productivity than passive diagnosis or alternating
active diagnosis with production.
The integration of diagnostic goals in the production
strategy results in informative production. The primary
objective in informative production is to continue pro-
duction. Under the assumption that there are various
ways to achieve the production goals, informative pro-
duction simultaneously maximizes diagnostic informa-
tion. The literature describes different types of pro-
duction such as simple production, time efficient pro-
duction, cost efficient production and, robust produc-
tion. All of those share the primary objective of achiev-
ing production but differ in the way they approach the
goal. In simple production any strategy that achieves
the production goal qualifies. In all other approaches
the set of production strategies, those which achieve the
production goals, are ranked by a secondary objective
function and the best production strategy dominates.
For example in time efficient production, strategies are
ranked by cost and the most cost efficient production
strategy dominates. Similar to other production strate-
gies informative production ranks the set of plans that
achieve production goals by their potential information
gain and selects the most promising strategy.

4 System

As part of our group work on self-aware, planner-driven
systems [8] we have designed and built the modular re-
dundant printing engines illustrated in Figure 1. The
system is controlled by a model-based planner [5]. The
model of the systems describes all the components in
the system, the connections between the components
and all the actions a component can take. The task of
the planner is to find the sequence of actions, called
plan, which will move sheets through the system to gen-
erate the requested output.
Expanding this work on model-based controlled system
we integrate a framework which integrates planning and
diagnosis to optimize production for long-run produc-
tivity called ‘Pervasive Diagnosis’.

Figure 1: Model of PARC’s prototype highly redundant
printer. It consists of two towers each containing 2 print-
ers (large rectangles). Sheets enters on the left and exit
on the right. Dark black edges with small rollers repre-
sents a possible paper path. There are three main paper
(horizontal) highways within the fixture. The fixture incor-
porates 2 types of media handling modules represented by
small lighter edge rectangles. The motivation for this design
is to continue printing even if some of the print engines fail
or some of the paper handling modules fail or jam.

Figure 2: Overall System Architecture

The overall framework of the system is depicted
schematically in Figure 2. The planner creates an infor-
mative production plan that achieves production goals
and generates informative observations. This plan is
then executed on the system. For many domains, a
plan consisting of numerous actions must be executed
before a useful observation can be made. The diagnos-
tic engine updates its beliefs to be consistent with the
plan executed by the system and the observations. The
diagnostic engine forwards updated beliefs to the plan-
ner to inform the planner about the new information
needs of the system. The planner and diagnosis engine
both run in real time.
5 Representing Information

In order to develop an algorithm to find plans efficiently, we will need to be more precise about what a system is, what the effect of a production plan on the system is, and what makes a plan informative.

We model a system as a state machine with actions $A$. The system is controlled by plan $p$ consisting of a sequence of actions $a_1, a_2, \ldots, a_n$ drawn from $A$. Executing an action potentially changes the system state. Part of the system state may be used to represent the state of a product. In this way, execution of actions can also contribute to creating a product. The internal constraints of the system limit the set of feasible plans to a subset of all possible sequences. From the point of view of diagnosis, executing the actions of plan $p$ results in a single observable, the plan outcome or observation $O$. In keeping with the diagnosis literature, we define two outcomes: the abnormal outcome, denoted $ab(p)$, in which the plan fails to achieve its production goal and the not abnormal outcome, denoted $\neg ab(p)$, in which the plan does achieve the production goal.

Information about the system is represented by the diagnosis engine’s belief in various possible hypotheses. A hypothesis is an assignment of abnormality or not abnormal to each of the system actions $e.g.$, $h = [ab(a_1), \neg ab(a_2), \ldots, ab(a_n)]$. In the single fault case, exactly one action will be abnormal. Let $H_{sys}$ be the set of all hypotheses. We distinguish one special hypothesis: the no fault hypothesis, $h_0$ under which all actions are not abnormal. Since every hypothesis is a complete assignment of abnormality to each action, they are all unique and mutually exclusive (e.g., $\forall h_1, h_2 \in H_{sys}, h_1 \neq h_2$.)

The system’s beliefs are represented by a probability distribution over the hypothesis space $H_{sys}$, $Pr(H)$. The beliefs are updated by a diagnosis engine from past observations using Bayes’ rule to get a posterior distribution over the unknown hypothesis $H$ given observation $O$ and plan $P$:

$$Pr(H|O, P) = \frac{Pr(O|H, P)Pr(H)}{Pr(O)}$$

The probability update is addressed in more detail in a related forthcoming paper. For this paper we simply assume that we have a diagnosis engine that maintains the distribution for us.

A plan is said to be informative, if it contributes information to the diagnosis engine’s beliefs. We can measure this formally as the mutual information between the system beliefs $Pr(H)$ and the plan outcome conditioned on the plan executed, $I(H;O|P = p)$. The mutual information is defined in terms of entropy or uncertainty implied by a probability distribution. A uniform distribution has high uncertainty and a deterministic one low uncertainty. An informative plan reduces the uncertainty of the system’s beliefs. Intuitively, plans with outcomes that are hard to predict are the more informative. If we know a plan will succeed with certainty, we learn nothing by executing it. In a forthcoming paper, we explain how to calculate the optimal amount of uncertainty $T$ a diagnosis engine should seek in a diagnosis plan in order to maximize information. In the case of persistent faults, the optimal uncertainty about the outcome would be $T = 0.5$. In the intermittent case, the uncertainty should lie in the range $0.36 \leq T \leq 0.5$. In this paper, we focus on the problem of finding a plan with a given amount of uncertainty $T$.

The first task is to be able to predict the uncertainty associated with a given plan $p = [a_1, a_2, \ldots, a_n]$. We denote the set of unique actions in a plan $A_p = \cup_i \{a_i \in p\}$. We assume catastrophic failures, meaning a plan will be abnormal $ab(p)$ if one or more of its actions are abnormal:

$$ab(a_1) \land \cdots \land ab(a_n) \Rightarrow ab(p) \text{ for } a_i \in A_p$$

The predicted probability of a plan action being abnormal will be a function of the probability assigned to all relevant hypotheses. The set of hypotheses that bear on the uncertainty of the outcome of plan $p$ is denoted $H_p$ and is defined as:

$$H_p = \{h| h \in H_{sys} \land h \Rightarrow ab(a), a \in A_p\}$$

Given a distribution over hypotheses and the set $H_p$ of explanatory hypotheses for plan $p$, it is possible to calculate the probability that plan $p$ will fail. Since every hypothesis $h \in H_p$ contains at least one abnormal action that is also in plan $p$, hypothesis $h$ being true implies $ab(p)$:

$$(h_1 \lor h_2 \lor \cdots \lor h_m) \Leftrightarrow ab(p) \text{ where } h_i \in H_p$$

Since the hypotheses are mutually exclusive by definition, the probability of a plan failure $Pr(ab(p))$ is defined as the sum of all probabilities of hypotheses which imply the plan to fail:

$$Pr(ab(p)) = \sum_{h \in H_p} Pr(h)$$

5.1 Search for Highly Diagnostic Plans

Our goal now is to find a plan which achieves production goals, but is also informative: that is, the plan is observed to have an abnormal outcome with probability $T$. Intentionally choosing a plan with a positive probability of failure lowers our immediate production, but the information gained allows us to maximize future production over a longer time horizon. A naive approach search would generate all possible sequences and filter this list to get the plans that achieve goals and are informative.

$$p^{opt} = \arg\min_{p \in \mathcal{P}} \{Pr(ab(p)) - T\}$$

Unfortunately the space of plans $\mathcal{P}$ is typically exponential in plan length. We can reduce this by considering families of plans that share structure. In $A^*$ planning, there are a set of partial plans $p_1 \rightarrow S_1, p_1 \rightarrow S_2, \ldots, p_1 \rightarrow S_n$. These plans go from the initial state $I$ to the intermediate states $S_1, S_2, \ldots, S_n$. At each step, $A^*$ tries to expand the plan most likely to achieve the goal in the best way. The ideal plan $p$ would start with the prefix $p_1 \rightarrow S_n$ which takes us to state $S_n$ and continue with the suffix plan $p_{S_n} \rightarrow G$ leading from the state $S_n$ to the goal state $G$. $A^*$ chooses the partial plan $p_1 \rightarrow S_n$ to expand
using a heuristic function \( f(S_n) \). The heuristic function \( f(S_n) \) estimates the total path quality as the quality of the plan prefix \( p_{1 \rightarrow S_n} \), traditionally written \( g(S_n) \), plus the predicted quality of the suffix \( p_{S_n \rightarrow G} \), traditionally written \( h(S_n) \).

\[
f(S_n) = g(S_n) + h(S_n).
\]

If \( f(S_n) \) never overestimates the true quality of the complete plan, then \( f(S_n) \) is said to be admissible and A* is guaranteed to return an optimal plan. The underestimation causes A* to be optimistic in the face of uncertainty ensuring that uncertain plans are explored before committing to completed plans known to be high in quality. The more accurate the heuristic function, the more A* will focus its search only on the high quality plans.

We want the search to run in real time online. We use an idea from the search community to automatically construct a good heuristic function from the description the system architecture and dynamics. Consider the example in Figure 3. In this example, the graph represents legal action sequences or possible plans that can be executed on the system. The system starts in the start state labeled \( S \) and follows the arcs through the graph to reach the goal state \( G \).

![Graph representation of system states and actions](image)

Figure 3: Machine topology places limits on what can be learned in the suffix of a plan.

Suppose that we execute a plan that moves the system through the state sequence \([S, A, C, G]\). The sequence is shown as a shaded background on the relevant links in the figure. We assume the plan resulted in an abnormal outcome. Unknown to the diagnosis engine, the abnormal outcome was caused by action \( a_{A,C} \). To keep things simple, imagine there is only this single, persistent fault. A diagnosis engine would now suspect all of the actions along the plan path to be faulty. We would have three possible probability hypotheses corresponding to suspected actions: \( \{a_{S,A}\}, \{a_{A,C}\}, \{a_{C,G}\} \). In the absence of additional information, we would assign them equal probability (see Table 1).

The graph structure and probability estimates can be used to construct heuristic bounds on the uncertainty that can be contributed to a plan by any plan suffix. We build up the heuristic from the end of the plan (right side of figure). Consider action \( a_{D,G} \) leading from state \( D \) to the goal state \( G \) in Figure 3. Action \( a_{D,G} \) was not part of the plan that was observed to fail, so it is not a candidate hypothesis. Under the single fault hypothesis it has probability zero of being faulted. If we extend any prefix plan ending in state \( D \) with action \( a_{D,G} \), we don’t increase the failure probability of the extended plan, because the action \( a_{D,G} \) has probability zero of being abnormal. There are no other possible plans from \( D \) to \( G \) so both the upper and lower bound for any plan ending in state \( D \) is zero.

Similarly, we determine that state \( B \) also has a lower bound of zero since it can be completed by an action \( a_{B,D} \) which does not use a suspected action and ends in state \( D \) which has lower bound zero. State \( B \) has an upper bound of 1/3 since it can be completed by an unsuspected action \( a_{B,C} \) to state \( C \) which has both an upper and lower bound of 1/3 probability of being abnormal.

Once we have recursively built up bounds on the probability of a suffix being abnormal, we can use these bounds with a forward search for a plan that achieves the target probability \( T \). Consider a plan starting with action \( a_{S,A} \). Action \( a_{S,A} \) was part of the plan that was observed to be abnormal. If we add \( a_{S,A} \) to a partial plan, it must add 1/3 probability to the chance of failure as it is a candidate itself. After \( a_{S,A} \) the system would be in state \( A \). A plan could be completed through \( D \). Action \( a_{A,D} \) itself, has zero probability of being abnormal, and using our heuristic bound, we know a completion through \( D \) must add exactly zero probability of being abnormal. Alternatively, from node \( A \), a plan could also be completed through node \( C \). Action \( a_{A,C} \) immediately adds 1/3 probability of failure to our plan and using our heuristic bound, we know the completion through \( C \) must add exactly 1/3 probability of being abnormal to a plan. The precomputed heuristic therefore allows us to predict total plan abnormality probability. The lower bound of the total plan is 1/3. This comes from 1/3 from \( a_{S,A} \) plus 0 from the completion \( a_{A,D}, a_{D,G} \). The upper bound is 3/3 equal to the sum of 1/3 from \( a_{S,A} \) plus 1/3 from \( a_{A,C} \) and 1/3 from \( a_{C,G} \).

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>( a_{S,A} )</th>
<th>( a_{A,C} )</th>
<th>( a_{C,G} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>1/3</td>
<td>1/3</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Table 1: Probability of Hypotheses (single fault)
from all goal states. A goal state has an empty set of suffix plans \( P_G = \emptyset \) therefore we set the lower bound \( L_G = 0 \) and upper bound \( U_G = 0 \). Each new state \( S_m \) calculates its bounds based on the bounds of all possible successor states \( SIC(S_m) \) and on the failure probability of the connecting action \( a_{S_m-S_n} \) between \( S_m \) and \( S_n \). A successor state \( S_n \) of a state \( S_m \) is any state that can be reached in a single step starting from node \( S_m \). In the single fault case it is true that the failure probability added to a plan \( p_{1-S_n} \) by concatenating an action \( a_{S_m-S_n} \) is independent from the plan \( p_{1-S_n} \) if \( H_{1-S_m} \cap H_{S_m-S_n} = \emptyset \). The lower bound \( L_m \) will be determined by the action probabilities linking \( S_m \)’s to its immediate successors and the lower bounds on these successors. The upper bounds are analogous.

\[
L_{S_m} = \min_{S_n \in SIC(S_m)} [\Pr(ab(a_{S_m-S_n})) + L_{S_n}]
\]

\[
U_{S_m} = \max_{S_n \in SIC(S_m)} [\Pr(ab(a_{S_m-S_n})) + U_{S_n}]
\]

In contrast to the computation of the heuristic, the search for an informative production plan starts from the initial state \( S \) and works recursively forward. The abnormality probability of the empty plan starting at state \( S \) is zero plus the best completion. In general, the abnormality probability will be the abnormality probability of the plan up to the current state plus the abnormality probability of the best completion. Since we are uncertain about the completion, its probability of abnormality will be an interval composed of a lower and upper bound. This makes the total abnormality probability an interval as well:

\[
I(p_{1-S_n}) = [\Pr(ab(p_{1-S_n})) + L_{S_n}, \Pr(ab(p_{1-S_n})) + U_{S_n}]
\]  

(7)

Recall, the best plan is the one whose total failure probability is \( T \). In the persistent fault case, \( T = 0.5 \). Given an interval describing bounds on the total abnormality probability of a plan, \( I(p_{1-S_n}) \) we can construct an interval describing on how close the abnormality probabilities will be to \( T \):

\[
|T - I(p_{1-S_n})|
\]  

(8)

The absolute value folds the range around \( T \). If the estimated total abnormality probability of the plan straddles target probability \( T \), then the interval \( |T - I(p_{1-S_n})| \) straddles zero and the interval will range from zero to the absolute max of \( I(p_{1-S_n}) \).

Let us define our search heuristic \( F(p_{1-S_n}) = \min(|T - I(p_{1-S_n})|) \). The function \( F \) has some interesting properties: Whenever the predicted total plan abnormality probability lies between \( L \) and \( U \), \( F \) is zero. Intuitively, it is possible that there exists whose abnormality probability exactly achieves probability \( T \). In all cases \( F(p_{1-S_n}) \) represents the closest any plan going through state \( S_n \) can come to the target abnormality probability exactly \( T \).

In our search we have a whole set of partial plans \( P = \{p_{1-S_1}, p_{1-S_2}, \ldots, p_{1-S_n}\} \). For each partial plan, we evaluate \( F(p_{1-S_n}) \) and expand the plan with the lowest value. Since \( F(p_{1-S_n}) \) is an underestimate, \(^*\) search using this estimate will return the most informative plan that achieves production goals.

5.2 Improving Efficiency by Search Space Pruning

In the previous section we have described an \(^*\) like search algorithm to find highly informative diagnostic plans. In many cases the search heuristic returns the same value, namely zero. This leads to little guidance. In this section we introduce several methods to focus the search.

The first method prunes out dominated parts of the search space. Consider an abnormality probability interval for a partial plan \( I(p_{1-S_n}) \) that does not straddle the target value \( T \). The best possible plan in this interval will be on one of the two boundaries of the interval, whichever one lies closest to the target value \( T \). Let \( L_I(p_{1-S_n}) \) and \( U_I(p_{1-S_n}) \) be the lower and upper bound of the abnormality probability interval \( I(p_{1-S_n}) \). Then

\[
V_{p_{1-S_n}} = \min(|L_I(p_{1-S_n}) - T|, |U_I(p_{1-S_n}) - T|)
\]  

(9)

will be the value of the best plan in the interval. Plan \( p_{1-S_n} \) will dominate every plan \( p_{1-S_j} \) where \( V_{p_{1-S_j}} < V_{p_{1-S_i}} \) and \( T \notin I(p_{1-S_j}) \). We can prune out all dominated plans from the \(^*\) search frontier.

The next method is used to intelligently break ties in the heuristic value. The heuristic value determines which node will be expanded next. It is possible that two or more nodes will receive the same value so that we might need a tie-breaking rule to decide which node should be expanded first. The simplest tie-breaking rule would be to pick a node randomly, but we can do better than that.

\( V_{p_{1-S_n}} \) represents a guaranteed lower bound on a total plan \( p_{1-G} \) starting with the partial plan \( p_{1-S_n} \) as prefix. Recall, the upper and lower bounds are realizable, but none of the interior points of the interval are guaranteed to exist. Therefore comparing the \( V \)'s enables us to decide which of two partial plans has the closest realizable solution. If two partial plans are also identical in this parameter, the information gain is the same, then we prefer the partial plan with less likelihood to fail. We put these two ideas together in a sequential decision procedure. Let \( p_{1-S_1} \) and \( p_{1-S_2} \) be the two partial plans with the same minimum value, i.e. \( F(p_{1-S_1}) = F(p_{1-S_2}) \). Then we break the tie by choosing the first rule that applies from the following ordered list:

1. If \( V_{p_{1-S_1}} < V_{p_{1-S_2}} \) then expand \( p_{1-S_1} \) first
2. If \( V_{p_{1-S_2}} > V_{p_{1-S_1}} \) then expand \( p_{1-S_1} \) first
3. If \( U_I(p_{1-S_1}) < U_I(p_{1-S_2}) \) then expand \( p_{1-S_1} \) first
4. If \( U_I(p_{1-S_1}) > U_I(p_{1-S_2}) \) then expand \( p_{1-S_2} \) first
5. If \( L_I(p_{1-S_1}) < L_I(p_{1-S_2}) \) then expand \( p_{1-S_1} \) first
6. If \( L_I(p_{1-S_1}) > L_I(p_{1-S_2}) \) then expand \( p_{1-S_2} \) first
7. otherwise pick randomly

6 Experiments

To evaluate the practical benefits of pervasive diagnosis, we implemented the heuristic search. We combined

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
it with an existing model-based planner and diagnosis engine and tested the combined system on a model of a modular digital printing press [16] or [6]. Multiple pathways allow the system to parallelize production, use specialized print engines for specific sheets (spot color) and reroute around failed modules. A schematic diagram showing the paper paths in the machine appears in Figure 4.

We do a test run for each possible abnormal action. The planner then receives a job from the queue. It then sends the plan to a simulation of the printing press. The simulation models the physical dynamics of the paper moving through the system. Plans that execute on this simulation will execute unmodified on our physical prototype machines in the laboratory. The simulation determines the outcome of the job. If the job is completed without any dog ears (bent corners) or wrinkles and deposited in the requested finisher tray, we say the plan succeeded, or in the language of diagnosis, the plan was not abnormal, otherwise the plan was abnormal.

The original plan and the outcome of executing the plan are sent to the diagnosis engine. The engine updates the hypothesis probabilities. When a fault occurs, the planner greedily searches for the most informative plan. Since there is a delay between submitting a plan and receiving the outcome, we plan production jobs from the job queue without optimizing for information gain until the outcome is returned. This keeps productivity high.

We evaluate performance of passive diagnosis (only normal operation), explicit diagnosis (alternates diagnosis and regular operation) and pervasive diagnosis (regular operation modified to obtain additional diagnostic information). In the case of explicit diagnosis, the planner solely focuses on the needs of the diagnosis engine and thus creates plans that maximize information gain in regards to the fault hypotheses. In general these plans need not be production plans. In case of passive diagnosis, the planner is not influenced by the diagnosis engine at all, that is, plans will be solely optimized in regards to production. In pervasive diagnosis, the plan is biased to have an outcome probability closest to the target $T$. As shown in Figure 5, the bias can create paths capable of isolating faults in specific actions.

In our model we assume, that under normal conditions the machine produces output at its nominal rate $r_{nom}$ and that diagnosis efforts begin once some abnormal outcome is observed (i.e. a paper jam, dog ear, etc.). This outcome is the result of some single faulty module within the system. If this module contributes to some plan, it will exhibit aberrant behavior with some probability $q$, the so called intermittency rate, resulting in plan failure. In short we assume single, intermittent, catastrophic faults. Here we use a simple cost model of opportunity costs in terms of unrealized production due to efforts of isolating the faulty component (diagnosis costs) and exchanging this component (repair costs). The cost in this model represent the expected total amount of lost production due to the fault.

The repair cost model is rather simple: The technician receives a list of the printer modules in decreasing order of their fault probability. He or she will then follow a very simple procedure: He or she will step through the list, exchange the next module and test the machine until this test shows the machine working properly. We assume each step takes a constant amount of time and thus a constant cost. Using this model, repair costs can easily be estimated by Equation 10, with exchange cost $c_{exc} = r_{nom} \times t_{exc}$.

$$c_{rep} = c_{exc} \sum_{i=1}^{\lfloor p^t \rfloor} i \cdot p_i^t$$

The cost model for the response cost depends on the diagnosis cost model and the repair cost model as shown in Equation 11.

$$c_{response} = c_{rep} + c_{policy}$$

At the moment the unexpected behavior is first observed the cost is 1 (lost unit of production) and the belief state is an uniform distribution over all fault hypothesis for all approaches. With the passive approach, the machine will continue to produce at rate $r_{nom}$, so diagnostic cost consists solely of the faulty output as shown in Equation 12.

$$c_{pass} = 1 + n_{faulty}$$

For explicit diagnosis, we assume no production during diagnosis (reasonable as chances of producing the right
output are negligible), thus diagnosis cost is calculated by Equation 13.

\[ c^\text{diag}_{c,t} = 1 + r_{\text{nom}} \times t_{\text{diag}} \] (13)

Finally for pervasive diagnosis we assume that the machine produces at a reduced rate \( r_{\text{perv}} \leq r_{\text{diag}} \) during diagnosis. Furthermore products can be faulty during production, thus the cost is calculated by Equation 14. Note that in our model the response cost represent the production lost due to the fault and therefore the goal is to minimize response cost.

\[ c^\text{perv}_{c,t} = 1 + n_{\text{faulty}} + (r_{\text{nom}} - r_{\text{perv}}) \times t_{\text{diag}} \] (14)

In our experiments we set the exchange time for a single module to 10 minutes. The exchange of a single module causes us to halt only one print engine at the time. Due to the fact that the underlying system operates with four print engines we set \( t_{\text{exc}} = 150 \text{sec.} \). The nominal rate of the system is \( r_{\text{nom}} = 3.1 \text{sheets/sec.} \). Our experiments have shown that the reduced rate of pervasive diagnosis is \( r_{\text{perv}} = 1.9\text{sheets/sec.} \). Based on the introduced model we compare the performance of passive diagnosis, explicit diagnosis and pervasive diagnosis for three levels of fault intermittency represented by the probability \( q \). When \( q = 1 \), a faulty action always causes the plan to fail. When \( q = 0.01 \) a faulty action only causes the plan to fail with a statistical mean of \( 1/100 \). The summary of the experimental results are in Table 2. A more detailed visualization of the results is presented in Figure 6 for the intermittency rate \( q = 0.01 \), in Figure 7 for the intermittency rate \( q = 0.1 \), and in Figure 8 for the intermittency rate \( q = 1 \). We averaged experiments over 100 runs to reduce statistical variation.

### Table 2: Pervasive diagnosis has the lowest rate of lost production. (# of exch. mod.: expected # of exchanged modules at the minimal response cost)

<table>
<thead>
<tr>
<th>( q )</th>
<th>min. response cost</th>
<th>Time</th>
<th># of exch. mod.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1010.22</td>
<td>214.1</td>
<td>2.01</td>
</tr>
<tr>
<td></td>
<td>Explicit</td>
<td>947.25</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>Pervasive</td>
<td>768.80</td>
<td>204.6</td>
</tr>
<tr>
<td>0.1</td>
<td>1055.25</td>
<td>35.0</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td>Explicit</td>
<td>591.31</td>
<td>29.1</td>
</tr>
<tr>
<td></td>
<td>Pervasive</td>
<td>547.77</td>
<td>37.2</td>
</tr>
<tr>
<td>1.0</td>
<td>1012.00</td>
<td>7.78</td>
<td>2.01</td>
</tr>
<tr>
<td></td>
<td>Explicit</td>
<td>615.43</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td>Pervasive</td>
<td>509.76</td>
<td>3.78</td>
</tr>
</tbody>
</table>

Table 6, 7, and 8) show the expected costs of repair, the cost of diagnosis and their sum over time in terms of lost production in relation to a healthy machine. We plot these costs for fault intermittency rates of 1, 0.1 and 0.01. Cost of repair is computed by estimating the repair time based on the current probability distribution over the fault hypothesis and pricing this downtime according to the nominal machine production rate. Cost

Figure 6: Experimental results for the intermittent rate \( q = 0.01 \) averaged over 100 runs.

Figure 7: Experimental results for the intermittent rate \( q = 0.1 \) averaged over 100 runs.

Figure 8: Experimental results for the intermittent rate \( q = 1 \) averaged over 100 runs.
of diagnosis at time $t$ is the accumulated production deficit in relation to a healthy machine producing at its nominal rate $r_{nom}$. The $x$-axis is the amount of time (relative to the first occurrence of the fault) after which one chooses to stop diagnosis and start repairing the machine. The minimum of the sum these costs denotes the optimal point in time (relative to first occurrence of the fault) to switch from diagnosis to repair and gives the minimal expected total loss of production due to the fault.

In all our experiments we found the optimal response costs of pervasive diagnosis to be below those of the other two approaches. As expected the respective optimal durations of diagnosis processes are in the order (shortest to longest) of explicit, pervasive and passive. This corresponds to the fact that explicit diagnosis focuses solely on the diagnosis task. Therefore explicit diagnosis is able to select plans with maximal diagnosis information gain and can isolate the faulty component in the shortest amount of time. However, due to the high production loss (production is halted), explicit diagnosis does not result in minimal response costs. Passive diagnosis has the lowest rate of lost production, but incurs the highest expected repair costs due to its lower quality diagnosis. This corresponds to the fact that the plans executed during passive diagnosis are optimized for production regardless of diagnosis needs. Pervasive diagnosis intelligently integrates diagnosis goals into production plans by using planning flexibility. This leads to a lower minimal total expected production loss in comparison to passive and explicit diagnosis.

7 Discussion

In the body of this paper we presented an application of pervasive diagnosis to a printing domain. We believe the technique generalizes to a wide class of production manufacturing problems in which it is important to optimize efficiency but the cost of failure for any one job is low compared to stopping the production system to perform explicit diagnosis.

The application that we have developed illustrates a single fault, single appearance, independent fault instantiation of the pervasive diagnosis framework. To generalize the instantiation we can reduce the set of assumptions by incorporating a different representation of the hypotheses space, the belief model and the belief update. In the most general case, multiple intermittent faults with multiple action appearance, the construction of the heuristic directly extends to an on-line forward heuristic computation similar to the one used in the FF planning system [9]. However, the idea of pervasive diagnosis is not limited to a probability based $A^*$ search. Another possibility of instantiation could be an SAT-solver approach in which the clauses represent failed plans and each satisfying assignment is interpreted as a valid diagnosis.

8 Conclusions

The idea of Pervasive Diagnosis opens up new opportunities to efficiently exploit diagnostic information for the optimization of the throughput of model-based systems. Hard to diagnose intermittent faults which would have required expensive production stoppages can now be addressed on line during production. While pervasive diagnosis has interesting theoretical advantages, we have shown that a combination of heuristic planning and classical diagnosis can be used to create practical real time applications as well.

References


Why Model Based Diagnosis for Spacecraft Never Took Off

James Kurien
NASA Ames Research Center
MS 269-3, Moffett Field, CA 94035
e-mail: James.A.Kurien@nasa.gov

María Dolores R-Moreno
Departamento de Automática.
Universidad de Alcalá.
Ctra. Madrid-Barcelona, km 33,6.
28805 Alcalá de Henares (Madrid), Spain.
e-mail: mdelores@aut.uah.es

Abstract
Robotic spacecraft appear to be the killer application for model-based diagnosis and recovery. They are complex, expensive assets that must be protected from loss due to failure. Fault protection, the task of manually designing, implementing and operating spacecraft systems that are robust in the face of anomalies, is time consuming and costly. Thus the mid-1990’s began a significant push to develop model-based diagnosis and recovery for robotic spacecraft, including flying two systems as experiments on spacecraft. However, to our knowledge no spacecraft in the subsequent decade has used any model-based diagnosis technology in operations and the expected benefits have yet to appear. Using examples from real spacecraft missions we probe the cost, benefit and risk advantages we anticipated. We identify the factors determining the expected value provided by a diagnosis and recovery system. We discuss the expected value such a system would provide an existing mission given the actual anomalies it experienced. This reveals the incorrect assumptions behind our expectations for model-based diagnosis in this domain.

1 Introduction
Robotic spacecraft appear to be the killer application for model-based diagnosis and recovery. They are complex and very expensive assets that must be protected from loss due to failure. Due to the nature of deep space they are out of reach and out of communication for extended periods of time. Fault protection, the task of manually designing, implementing and operating spacecraft systems that are robust in the face of anomalies, is time consuming and costly. Thus the mid-1990’s began a significant push to develop model-based diagnosis and recovery for robotic spacecraft. A great deal of experience was gained applying two such diagnosis systems to simulated or tested domains involving spacecraft, chemical processors, life support systems and scientific instruments, and demonstrating them during flight experiments on the Deep Space 1 (DS-1) and Earth Observer 1 (EO-1) spacecraft. However to our knowledge no spacecraft project in the decade since has adopted these or any other model-based diagnosis (MBD) technologies in operations and the anticipated benefits have yet to come to fruition. This suggests a number of questions. What are the costs of using MBD, what benefits are observed in practice and how can we approach the question of whether the benefits outweigh the costs for spacecraft? How are missions today approaching fault diagnosis and recovery during operations? How does the proposed MBD approach differ with the current practice, and why aren’t we able to leverage MBD technologies to provide greater value or lower cost? By attempting to answer these questions, we have identified specific assumptions that made our expectations for MBD in this domain incorrect.

In the next section, we describe the spacecraft fault protection problem and common practice for addressing it, followed by an overview of the Livingstone model-based diagnosis and recovery system. We then consider Livingstone’s impact on spacecraft fault protection practice, both in terms of the impact we expected based on its capabilities and the impact we have observed in the subsequent years. In the next sections, we discuss the pressures on expected value, cost and risk we believe hamper the use of on-board MBD and recovery systems. We support our analysis using the MER and D-1 missions as examples. In the Discussion we summarize our experience with the cost, risk and value tradeoff for MBD and our understanding of why these technologies have not been adopted in the spacecraft domain. In Related Work, rather than survey the MDB field, we briefly discuss a paper that raised similar questions about MBD and a few non-MBD systems used on spacecraft.

2 Traditional Fault Protection Process
Before discussing MBD we briefly discuss spacecraft Fault Protection (FP) for the purpose of comparison. The primary purpose of FP is to ensure that anomalies or operational problems encountered during operation of the spacecraft do not result in permanent reduction in the spacecraft’s capabilities or loss of the mission itself. As Neilson explains in an excellent overview of the FP system for the Mars Exploration Rovers (MER), FP is an engineering process that incorporates robustness to faults into spacecraft hardware, software, systems engineering and operations [Neilson2005]. All of these systems are engineered to work together to reduce the likelihood that any reasonably plausible contingency will result in
permanent loss of mission capabilities. This paper is largely concerned with on-board software for active fault detection, isolation and recovery (FDIR). We note though that the on-board system is just one aspect of the overall FP engineering process, and that the on-board system for protecting the spacecraft is typically a mix of hardware and software. Traditionally, the FP engineering process is driven by fault modes, effects and criticality analysis (FMECA). This process typically determines the possible faults of a system or subsystem, some notion of their likelihoods, and an analysis of the impact of each. If the likelihood of a fault is deemed sufficiently high and the impact sufficiently negative, the analysis would also include how the fault would be detected and what the appropriate response would be.

The appropriate response to a fault may depend upon the phase of the mission. We use the term critical phase of a mission to mean periods when the spacecraft must take specific actions (a critical sequence) or serious degradation will result. For example during the entry sequence each MER rover entered the Martian atmosphere at over 10,000 miles per hour. At specific times it jettisoned a protective shell, deployed airbags, and the like. Failure to execute a step in this six minute sequence would end the mission. We say the system must fail operational in that any failure deemed sufficiently likely to occur must be taken into account by the FP approach, for example by switching between redundant subsystems to allow execution to continue. Accordingly, during development of the spacecraft an enormous amount of attention is given to the precise critical sequence the spacecraft will execute, which failures are likely, how they will be detected, and how they will be immediately mitigated. Since response must be timely, it must be available on-board. This may involve something as simple as a table mapping observed sensor values to commands that should be issued in response, for example to switch to a redundant backup. For very complex critical sequences on large spacecraft, a much more elaborate method of determining responses may be developed, such as for the Cassini orbital insertion at Saturn [Brown et al.1995].

In a non-critical phase, it is still possible to damage or lose the mission due to a fault, but there isn’t the added constraint of having to execute a critical sequence. Thus typical FP systems tend to focus on mitigating or postponing the impact of the fault by changing the spacecraft's state or behavior. Often if a serious problem is detected, the spacecraft is placed into a safe mode where the only actions taken are those that maintain the spacecraft in a quiescent state. For example on the MER rovers, draining the battery risks not being able to heat the rover during the cold Martian night and not being able to communicate with Earth when expected. The system level MER FP includes a hardware battery controller that disconnects the batteries should a serious hardware or software failure begin to drain them. In this case, the system’s heaters, solar panels, and flight software work together to ensure the rover operates safely and wakes up. During the day to contact Earth using very few hardware and software subsystems and minimal power. If the situation appears to be less grievous or more localized, then a less drastic response may be taken at first. On the MER rovers, subsystem behaviors incorporate subsystem level FP [Reeves2006]. If the rover’s arm draws more than the allowed current during use, it is marked failed. The arm behavior ignores any subsequent requests to use the arm, and the rover driving behavior is disabled if the arm is not stowed away. Routine communication with Earth and other tasks that do not involve the arm continue as normal. During the cycle of downlinking telemetry and uplinking sequences (typically the next day), ground operators can inspect the telemetry and debug the arm. In many cases, engineers may send commands to carefully gain more information about the spacecraft’s condition, or may attempt to reproduce the issue on an Earth-bound copy of the spacecraft. When the anomaly is understood, engineers may simply re-enable the spacecraft, upload a software patch, or change operational procedures to work around it. A process similar to this has protected the MER rovers for over four years [Matijevic and Dewell2006].

The above FP approach is to carefully engineer the robust but minimal fail operational capabilities needed for critical periods, and otherwise engineer hardware, software and operations so the spacecraft can be put in a safe state in response to plausible anomalies. This characterizes a wide range of FP systems that have been flown. The remainder of this paper is about experience gained attempting to improve upon this approach using model-based diagnosis technology.

3 The Livingstone System

Livingstone [Williams and Nayak1996, Kurien and Nayak2000] is a model-based diagnosis system in the style of GDE [de Kleer and Williams1987] and Sherlock [de Kleer and Williams1989]. A main innovation is the use of one system model and search algorithm to infer both what failures have put the system in its current state and what actions may be used to move the system to a state with a desired property. Thus it is capable of detecting, identifying and also responding to off-nominal conditions. With a model that includes actions, it can monitor execution of commands, detect anomalies, diagnose failures and provide recovery actions for complex systems such as an spacecraft.

Figure 1 illustrates its operation at an idealized, schematic level. We use the Cassini spacecraft as a benchmark example, as was the norm in the Livingstone papers. To avoid a persistent confusion, we note Livingstone was run against a simulation of the Cassini spacecraft and not as a part of the Cassini mission. Livingstone requires a high level model of

---

the components of the spacecraft and their operation. During operation, Livingstone is fed the stream of commands that are being given to the spacecraft and the readings from sensors that relay the internal state of the spacecraft (e.g. switch status bits, temperature sensors, pressure transducers, etc.). Livingstone uses the model of the spacecraft’s components and the command stream to predict the values of the sensors that should result from the commands assuming no components are failed. If there is a discrepancy between the predicted and observed sensors, then a failure is assumed. Livingstone uses the model of the components to simulate different combinations of failures. It uses a diagnosis algorithm to quickly focus on the combination of failures that would predict the sensor values that are being observed. This combination of component failures is the diagnosis.

In case of failure, Livingstone is able to use the same predictive model to suggest commands that achieve a desired property (e.g. engine is receiving fuel) and thus mitigate the failure. It can for example infer that one should switch between the failed component and a similar set of components that performs the same function. For example, consider the Cassini main propulsion system in Figure 2. The purpose of the system is for the helium tank to pressurize the oxidizer tank and fuel tank in order to push fuel and oxidizer to exactly one of the engines, where it is ignited to create thrust. Owing to Cassini’s decade long mission, the system has many redundant paths between the various tanks and a backup engine. Valves in parallel allow a second path in case a valve sticks shut. Valves in series allow a path to be turned off if a valve sticks open. When run against a simulation of this propulsion system, Livingstone’s task was to monitor the commands given to the valves and infer if the pressure readings were consistent with the expected state of the valves. If not, Livingstone was able to determine the smallest number of valves to open or close to ensure oxidizer and fuel could reach exactly one engine. Thus Livingstone could effectively manage the configuration of the system so that it always produced thrust when desired even in the face of failures.

4 Expected Benefits

The promise of a system like Livingstone flows from the concept that users describe the nominal and (perhaps multiple) failure behaviors of each type of component, and how components are connected. Livingstone then infers the system-wide behavior given any combination of nominal and failed components and any sequence of commands. Thus the user models that a valve can be open and allow fluid flow, or closed and stop fluid flow, or stuck open or stuck closed, and can arrange valve combinations far more complex than the Cassini model shown. Livingstone in turn performs system-wide reasoning during operations to start and stop fluid flow to the engines (or achieve any other state described by the model) in the face of multiple valves sticking closed or open. We envisioned a wide range of benefits if we could simply describe the local behavior of components instead of reasoning through system wide interactions under a multitude of scenarios.

- **Reduced fault protection implementation costs**
  Rather than develop a custom FP and safing system, the spacecraft would employ a re-usable diagnosis engine that inferred the spacecraft’s state and recovery responses on the fly from component models.

- **Significantly lower analysis costs for critical phases**
  The fail-operational response needed for a critical phase would be inferred from the model on-line during execution, reducing the manual analysis needed to engineer a response a priori and encode it.

- **Greater robustness than traditional fault protection**
  The on-board diagnoser would infer diagnoses of double and triple failures and suggest workarounds, where a traditional FP might include pre-computed responses for single failures or certain critical anomalies involving multiple failures.

- **Greater mission return**
  By inferring recoveries for any state of the spacecraft we could fail operational even in non-critical periods. Operations would resume with no need to wait for intervention from the ground, increasing spacecraft productivity.

- **Reduced mission staffing levels and operational costs**
  Since the MBD system would resolve many anomalies without intervention from the ground, fewer expert engineers would be needed during spacecraft operations.

5 Flight Experience

A large number of talented people developed Livingstone and L2 applications for a wide variety of systems, which allowed us to gain the experience upon which this paper is based. Table 1 lists applications developed for simulators, testbeds and two flight experiments, along with the year. A more detailed version of this table appears in [Kurien

<table>
<thead>
<tr>
<th>Type</th>
<th>Domain</th>
<th>Subsystem</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim</td>
<td>Cassini</td>
<td>Propulsion</td>
<td>Liv 1996</td>
</tr>
<tr>
<td>Flight</td>
<td>DS 1</td>
<td>Attitude control</td>
<td>Liv 1998</td>
</tr>
<tr>
<td>Testbed</td>
<td>ISRU</td>
<td>Chemical reactor</td>
<td>Liv 1998</td>
</tr>
<tr>
<td>Testbed</td>
<td>Interferometry</td>
<td>Optical bench</td>
<td>Liv 1999</td>
</tr>
<tr>
<td>Sim</td>
<td>μ-spacecraft</td>
<td>Drive system</td>
<td>Liv 2001</td>
</tr>
<tr>
<td>Sim</td>
<td>rover</td>
<td>Propulsion</td>
<td>L2 2002</td>
</tr>
<tr>
<td>Sim</td>
<td>X-34</td>
<td>Electronics</td>
<td>L2 2002</td>
</tr>
<tr>
<td>Flight</td>
<td>EO 1</td>
<td>Instruments</td>
<td>L2 2003</td>
</tr>
</tbody>
</table>

Figure 2: Cassini Main Propulsion System
and R-Moreno2008]. Livingstone was chosen in 1996 to be a part of the Remote Agent spacecraft autonomy architecture [Muscettola et al.1998]. In May 1999 it was flown on the Deep Space 1 spacecraft as a technology experiment [Bernard et al.1998]. The first author gained direct experience with model-based diagnosis technology by doing Livingstone development, modeling of the Deep Space 1 (DS-1) spacecraft, and participating in mission operations during the experiment. Livingstone was activated on DS-1 for a 20 hour test and a 6 hour test. During these tests, the Remote Agent ran on top of the DS-1 flight software, which included a fault protection system. During the test Livingstone was fed simulated sensor readings consistent with a set of four pre-determined failure scenarios: switch position indicator failed, camera power switch stuck on, science instrument not responding and thruster stuck closed. In the first case, Livingstone correctly ignored the sensor, and in the remaining cases recommended recoveries of re-trying the command, power-cycling the instrument, and switching thruster control modes, respectively.

Subsequently, we developed the follow-on Livingstone 2 system (L2), with several technical improvements [Kurien and Nayak2000]. With L2 and the associated modeling tools, our experience applying model-based diagnosis and recovery systems to spacecraft greatly expanded. L2 was flown as a technology experiment on the EO-1 spacecraft, again flying on top of the existing fault protection system. L2 was activated on EO-1 for a total of 143 days in 2004 and 2005 and diagnosed 13 simulated failures [Hayden and Sweet2004]. Flight experiments for the X-34 and X-37 spacecraft also were developed, though those vehicles were never flown [Maul et al.2004]. In addition to these flight experiments, Livingstone or L2 was applied to a Mars in-situ propellant production testbed (ISRU) at Kennedy Space Center [Larson and Goodrich2000], a testbed for the Space Interferometry Mission at the Jet Propulsion Laboratory, the Bio-Plex Mars habitat testbed at Johnson Space Center, a testbed for the PSA micro spacecraft [Dorais et al.2003], and a rover testbed [Bresina et al.1999].

6 Analysis of Impact

A great deal was learned from the efforts of the many talented teams that used Livingstone. In retrospect though, it’s fair to say that MBD did not transform, or even impact how fault protection is done on NASA missions. To our knowledge, no NASA spacecraft has used Livingstone or any system like it as a part of its fault protection system, and after the DS-1 and EO-1 experiments, there have yet to be additional research-funded flight experiments. This begs the question of why we did not observe the wide range of benefits we expected from applying model-based diagnosis and recovery to spacecraft operations, and why there was no ‘mission pull’ and adoption of the technology. It is tempting to explain the lack of penetration by imagining missions are averse to incorporating new technologies. Certainly revolutionary approaches are taken all the time. Consider the airbag landing system used on PATHFINDER and MER. Perhaps there is some hesitance to use unfamiliar or model-based software. Consider the case of planning and scheduling technology. Coincidentally, Livingstone flew with the HSTS planner during the Remote Agent experiment, and L2 flew with the CASPER planner on EO-1. At a high level, planners are similar to Livingstone in that they comprise a generic inference engine and a model used to adapt it to a problem. A planner chooses actions to take to achieve a goal, rather than failures to explain a symptom, but the algorithms and models are similar in the grand scheme of things. HSTS evolved into ground-based tools that have generated thousands of plans for the MER rovers and are used on the Phoenix Mars lander and upcoming Mars Science Lab rover [Aghelli et al.2006]. CASPER became an operational tool on EO-1, continuing to run on-board for years, plan over 100,000 goals to date, and save millions of dollars in operating costs [Chien et al.2003]. Thus missions are willing to adopt new, model-based software technologies.

As technology developers, there is a natural tendency to focus on the capabilities of a technology, such as MBD’s ability to find diagnoses and make recoveries. The decision by a mission to adopt a technology is a combination of the expected value, cost and risk given the characteristics of that particular mission. The three are intertwined. A mission may make a significant investment to buy down risk, as illustrated by the inclusion of a backup engine on Cassini. Conversely a mission feature that provides value may be abandoned if the cost cannot be kept under control. In these terms, we believe it’s relatively easy to understand the lack of penetration of MBD. We provide an overview before a detailed analysis.

Consider the non-critical phases of a mission and the proposal to add fail operational capability. The value of a diagnosis and recovery capability captured by our models is only realized if a failure occurs, it is correctly diagnosed and a recovery more productive than safing is achieved. Typically, spacecraft go into safe mode very rarely. Surveying real mission anomalies also reveals that few are failures that a priori we would have modeled and been able to recover. When failures do occur, operators typically need to carefully understand what has occurred and what assumptions about the spacecraft design have failed. They then upload commands to address the immediate situation and often make a change to operational rules for the spacecraft before proceeding. Thus we will argue the expected value provided by MBD during non-critical phases is not high, especially when considered against the cost and risk involved.

Consider the critical phases of a mission and the proposal to use MBD and recovery to provide the required fail operational capability. Typically a critical phase is minutes to hours long, involves a known sequence of events, and is critical in the fate of a mission costing hundreds of millions of dollars. The added value and reduced cost from employing MBD have not been clearly shown. However, they would have to be incredibly high to offset the risk of not being able to verify in advance exactly how the spacecraft was going to respond to plausible scenarios during this short portion of the mission when an incorrect response will result in mission loss. Thus we will argue the risk of MBD during critical phases is high, and the value is unclear. The expected net value of using MBD over the life of a mission depends upon the expected value that the technology provides, minus the additional risks of using it and the cost of deploying it. In the next sections,
we discuss the pressures on expected value, cost and risk we believe hamper the use of on-board MBD and recovery systems. We support our analysis using the MER and DS-1 missions as examples.

7 Low Expected Value

During Livingstone development we focused upon the promise that Livingstone could provide value to a mission, for example by potentially reducing the need for the spacecraft to enter a safe state. There was no attempt to quantify the magnitude of that value, leading us to overestimate it. In retrospect, in order for a MBD system to be exercised and provide value, a failure must occur, the MBD system must correctly diagnose the failure and it must suggest a useful response. It is the likelihood of these events that will determine usefulness of MBD. As with an insurance policy, the value actually provided by an MBD system during a mission isn’t known until after events have unfolded and the mission is complete. Unfortunately, we know of only two cases where an MBD system has been flown, which we discuss below. We then discuss methods for estimating the expected value if MBD were hypothetically deployed on a mission, and why we estimate it to be in general quite low.

During the two day Livingstone flight experiment and the 143 day L2 flight experiment, none of the modeled failures of the spacecraft occurred. Two failures in experiment-related software did occur during and interfere with the flight experiment of Livingstone on DS1. One failure we had not even considered modeling, and the other was explicitly declared out of scope during the modeling effort. Thus the only two actual failures known to have occurred during a Livingstone flight experiment were not identified or recovered from [Bernard et al. 1998]. The L2 flight experiment also generated 14 false positives, diagnosing modeled failures when none occurred. Thus in these two very short experiments, this particular MBD system provided no operational value.

These two data points involving a particular MBD technology are too few to draw any conclusions about the actual value provided by MBD for spacecraft applications. However, like an actuary, we can compute the expected value MBD would provide to other missions from the product of the likelihood of the events leading to MBD operating correctly and the value of the quantity. To estimate the expected value for a mission, we then need to estimate the following quantities:

- the likelihood of a failure actually occurring ($\leq 1$)
- the likelihood of correctly diagnosing the failure that actually occurs ($\leq 1$)
- the likelihood of providing proper response ($\leq 1$)
- the value of generating the correct diagnosis and response on-board (timeliness, reduced operations cost)

To provide a concrete example of how we would reason through this analysis for a mission, Table 2 summarizes Nelson’s excellent overview of the anomalies in MER operations that occurred in the first two years of commanding two rovers on Mars. This table captures the anomalies that caused the two rovers to enter a fault response during approximately 1550 days worth of operations. The first column is a short name we have given to the anomaly. The second denotes whether the problem was due to software, hardware or interactions with the environment. We presume that most MBD applications do not apply to dynamic interactions with the environment, though this is incidental to our analysis. The third column denotes how many days passed between when the anomaly occurred and the subsystem involved in the anomaly was put back in to routine operations. Note in many cases other non-affected subsystems were put back in to operation sooner. The final column describes what response was developed by the MER anomaly analysis team.

This table captures anomalies during non-critical portions of the mission. In this case, we define the ideal value of having on-board diagnosis and recovery to be equivalent to amount of operational time lost waiting in safe mode after an anomaly that could have used productively if the anomaly had been resolved immediately. A second term in expected value is the rate of failure within on-board spacecraft systems, which we expect to be quite low in general. This is important because the product of our value term and this rate, conceptually the expected amount of time the spacecraft spends in safe mode, is an upper bound on the value of on-board recovery, representing the ideal case where for every anomaly our MBD system produces a correct diagnosis and a productive recovery. This bound is simply the nature of attempting to provide value with on-board diagnosis and recovery.

Looking at Table 2 the small number of hardware and software related anomalies suggests that the rate of anomaly occurrence is quite small. The amount of time spent on anomaly recovery was about 2% of the operational time, including a significant amount of time responding to environmental problems such as being stuck in soft sand. Thus for MER we know the amount of potential operational time spent in safe mode turned out to be 2% of the operational time of the rover. This represents the ideal upper bound of the value of using MBD to achieve fail operational during routine operations if every anomaly could be correctly resolved autonomously on-board.

The reduction from ideal MBD performance to the expected value is determined by our estimates of the likelihood that a given MBD system will correctly diagnose these anomalies and the likelihood that it can provide a productive recovery. Looking at the failures in Table 2 the first question to consider is out of the enormous space of component failure modes, what is the likelihood that a priori our models would have covered diagnosis and recoveries for these failures? The second question is, what is the likelihood that our automated software would have produced recoveries that were both correct and more valuable than simply shutting down subsystems and waiting for ground analysis as the existing MER system does? Since MER did not use an on-board MBD system, discussing the likelihood of a correct diagnosis or recovery may be considered conjecture. We believe one can get a sense of the (small) magnitude of these likelihoods from columns 3 and 4 of the table. These reflect the amount of time the engineers who built the rover, flight software and operations system spent carefully probing the rovers with small experiments and deriving a recovery and new operational policy that would accommodate the failure and the rover’s interac-
tion with the environment. We believe that the MER pattern of infrequent, short bursts of extremely careful analysis indicates the effective value of model-based diagnosis would be significantly less than adding 2% to the rovers' operational time. In general, we believe the likelihood of failure is low for operational spacecraft, meaning even the potential benefit for MBD is limited. In addition, we consider the likelihood of correct diagnosis and recovery to be low, further lowering the expected value of MBD. Situations where combinatorics and propagation of information are the issue in generating diagnoses and recovery, as was initially suggested by Cassini’s 27 valve propulsion system and its 2^{27} configurations, do not appear to be the driving problem in spacecraft fault protection and anomaly resolution. For the types of applications we are familiar with, failures that do cause loss of operational time are typically complex, unexpected and break the modeling abstractions used to develop the system.

8 Non-Trivial Cost Increases

During Livingstone development, we projected future missions would enjoy substantial savings by eliminating the need to write a traditional fault protection system or to perform the system-level analysis done to determine the correct response to critical, mission ending anomalies. Much of the work previously done through system level analysis would be done through inference. Costs would largely consist of the effort to encode models for the MBD system, which would be reduced via graphical modeling tools developed with Livingstone.

We now believe that use of MBD adds significant analysis, development and testing costs. In retrospect, use of a model-based diagnosis system does not eliminate the need of a fault protection system, and thus does not eliminate that source of analysis costs. Creating the models required for an MBD system also requires significant analysis specific to this usage. Unlike traditional approaches, the value proposition of MBD also promotes performing a significant amount of “speculative” analysis for a range of non-critical failures that do not strictly need to be completely analyzed a priori. Finally, our experience is the envisioned MBD functionality adds a significant integration and testing effort to production of spacecraft flight software. We consider each factor in turn below.

We don’t expect use of MBD to reduce the need for analysis related to fault protection, because in retrospect diagnosis and fault protection are not equivalent. Fault protection is a system engineering process that impacts the design of hardware, software and operational procedures. It must ensure, for example, if any hardware, software, environmental or operational problem is draining the spacecraft’s batteries, the combined hardware, software, operations system and documented operator procedures have the maximum likelihood of stabilizing the situation before the vehicle is lost. This is a much broader problem than that of on-board component-level diagnosis and reconfiguration of the spacecraft. In addition, the ability to compute component-level diagnoses is often neither necessary nor sufficient to ensure spacecraft safety.

<table>
<thead>
<tr>
<th>Brief Description</th>
<th>Type</th>
<th>Analysis</th>
<th>Recovery process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel drive actuator</td>
<td>HW</td>
<td>4</td>
<td>Experiment to characterize capability of wheel</td>
</tr>
<tr>
<td>Steering current</td>
<td>HW</td>
<td>4</td>
<td>Experiment to determine source of current increase. K-turns to avoid steering failed wheel</td>
</tr>
<tr>
<td>Shoulder actuator current</td>
<td>HW</td>
<td>17</td>
<td>Experiment to characterize shoulder motor degradation. Characterization of future arm failure on driving. Change storage policy to minimize thermal cycling &amp; forestall failure</td>
</tr>
<tr>
<td>Heater stuck on</td>
<td>HW</td>
<td></td>
<td>Determine that survival heater was stuck on. Implement policy to remove batteries from power bus at night. Rely on solar power to wake rover at dawn. Trade off power savings vs. degradation of instrument due to cold. * Ops continued while problem was addressed</td>
</tr>
<tr>
<td>Late wakeup/dust storm</td>
<td>Env</td>
<td>1</td>
<td>Solar panels woke rover slightly late due to dust storm. Rover missed sequence start time then waited in standby mode. Plan new sequences to start 1 hour after expected wakeup</td>
</tr>
<tr>
<td>Rock in wheel</td>
<td>Env</td>
<td>7</td>
<td>Current spike explained by images of rock in wheel. Days of careful driving dislodge it.</td>
</tr>
<tr>
<td>Stuck in sand</td>
<td>Env</td>
<td>40</td>
<td>Imagery suggests rover is not moving, wheels 70% buried in sand</td>
</tr>
<tr>
<td>Flash file system anomaly</td>
<td>SW</td>
<td>14</td>
<td>Overloaded file system table prevents creation of new files. Rover continuously reboots. Under what on-board fault protection system is doing, what is causing reboots</td>
</tr>
<tr>
<td>Startup race condition</td>
<td>SW</td>
<td>2?</td>
<td>Lose comm window every few hundred sols. Added short keep out period after startup</td>
</tr>
<tr>
<td>Imaging race condition</td>
<td>SW</td>
<td>2</td>
<td>Imaging HW shut down while sequence still reading data from HW</td>
</tr>
<tr>
<td>Corrupt command</td>
<td>SW</td>
<td>6</td>
<td>Solar conjunction test of corrupt commands overloads command handler</td>
</tr>
<tr>
<td>Variable eval exception</td>
<td>SW</td>
<td>4</td>
<td>Same global defined in two sequences running in parallel, result in fault</td>
</tr>
<tr>
<td>Upload fault</td>
<td>SW</td>
<td>2</td>
<td>Initial uplink through orbiter experiment overloaded CPU. Pad uplink file, limit size</td>
</tr>
</tbody>
</table>

Table 2: MER Anomalies
The fault protection scheme typically identifies only faults (e.g., battery voltage is low) used to find a pre-planned response that is meant to safe the spacecraft or continue a very specific critical sequence over a huge space of problems induced by the hardware, software or environment. For example, the fault protection design might place all non-essential devices on a separate power bus, which is simply turned off if there is any power-related anomaly. Thus in the flight experiments, it was necessary to employ a fault protection system which did not need to do detailed diagnosis, while running Livingstone, which did not perform system fault protection. Since MBD does not typically address system fault protection, it has not been demonstrated (nor do we believe) that use of MBD technology eliminates or significantly reduces analysis costs for developing the basic fault protection capability missions require.

We imagined future missions might use the ability of model-based diagnosis to propagate behavior of individual components across a system model to generate recovery responses on the fly or before flight, meaning the analysis costs of a mission using MBD would drop. Two issues are where the models come from and what analysis do they eliminate? Model-based diagnosis requires a diagnostic model which is different than the simulation models used in routine spacecraft development. The DS-1 experience was that it’s necessary to know the aspects of each component relevant to failure, the plausible failures to be modeled, how they manifest themselves locally, local actions that can be taken, and so on in order to scope and write a model that can be used for anomaly detection, identification and recovery. Our experience was therefore that additional failure modes and effects analysis was needed to drive diagnostic modeling, rather than a model existing through some other process and then being used to replace traditional analysis.

The need for additional analysis is exacerbated by the proposition that an MBD system provide value by autonomously recovering from non-critical anomalies. Consider that the traditional fault protection strategy reserves the most detailed, a priori analysis only for critical sequences and the process of safing the spacecraft. The majority of possible faults simply trigger the safing system without diagnosis to the component level. Since further analysis is done post hoc, it’s done only those failures that actually occur, and without the need to codify the diagnosis into a model. The main value ascribed to MBD was that it would do detailed diagnosis and recovery autonomously. This means the analysis and modeling needed to diagnose and recover failures, and the non-trivial task of encoding those capabilities into software, must be done a priori, before we know which failures will occur. Thus we perform the detailed analysis and modeling needed to automatically recover for many failures most likely never occur, but are not necessary to automatically recover in the unlikely event they do.

An overlooked additional cost is integrating the model with the spacecraft and testing. Signals generated by the spacecraft’s internal sensors may need to be conditioned to remove transient disturbances or abstracted to match the diagnostic algorithm. Since we are attempting to use all available sensors to autonomously identify failures before abrupt, system-level effects are seen, the Livingstone experience is this can represent at least as much work as modeling. Thus we believe integration costs are higher than for fault protection systems that typically focus on anomaly detection without isolation and use a smaller number of system-level measures. With respect to testing, one of the main characteristics of Livingstone is the ability to generate combinations of diagnoses and recoveries from the possible diagnoses and recoveries for individual components. However, it was still necessary to work through the possible failures, how the failure would propagate through the system, and how Livingstone would respond, then validate the expected behavior through testing. Given the space of possible responses Livingstone could generate, the issue of how to validate it at a reasonable cost was an issue during experiments and remains an issue.

In summary, we believe the proposed use of MBD represents a cost increase rather than savings. The need to develop a separate set of component-level diagnostic models adds cost, and does not significantly offset fault protection analysis costs. The desire to have a priori, a system that can autonomously diagnose and recover a spacecraft even for non-critical anomalies appears to introduce additional detailed analysis, model encoding, signal conditioning and testing that is not necessary if the spacecraft is simply put in a safe mode when possible. We believe the approach is also at a cost disadvantage due to the more complex testing and verification needed to ensure any of the additional diagnoses and novel recoveries such a system might generate during autonomous operations would not endanger the spacecraft.

9 Increased Risk

Initially, we imagined that MBD would reduce risk of mission loss by generating diagnoses and recoveries on the fly and increasing the range of situations over which specialized fault responses were available. Missions considered that MBD would increase risk because of the increased complexity of the software’s response and our inability to concisely characterize, enumerate and validate the range of diagnoses and responses the system might undertake. Rather than engineer a small number of safing responses that are as broadly applicable as possible, MBD seeks to generate recovery responses that are as specialized as possible on the fly. It’s interesting to note that this does not necessarily imply that an MBD system responds to a broader range of anomalies, simply that it responds in a more specialized fashion to each. This means there are far more variations in spacecraft response based on its state, and the full set of conditions and responses could not be enumerated and tested. In addition, the purpose of MBD is to propagate information across the modeled system to allow variations in response. Thus it can be difficult to even concisely describe how small, non-local variations in the space of inputs will impact the response, and difficult to argue that a specific set of test cases provides good coverage for validation. Combating this perceived increase in risk is a challenge since the ability of MBD to respond with a far wider range of behaviors than traditional fault protection is both its selling point and the source of concern that the system will do something unpredictable. Some work has been done to ap-
ply model checking approaches [Cimatti et al. 2003], but this remains a significant issue for adoption.

An additional increase in risk comes from the proposal to continue to operate the spacecraft via a recovery generated on-board when it is possible to safe the spacecraft and await expert analysis. Consider the three wheel problems of Table 2, Wheel drive actuator, Rock stuck in wheel, and Stuck in sand. One can imagine mis-diagnosing which failure was occurring and applying the recovery meant for another (e.g., attempting to drive in circles when stuck in sand rather than when a rock is in the wheel) could permanently trap or damage the rover. We don’t have examples of Livingstone mis-diagnosing a spacecraft failure as no failures caused any diagnoses during the flight experiments. However, we do have examples of false positives (indicating a failure when none exists) during the EO-1 experiment. Thus it’s important to keep in mind that MBD may cause us to execute actions that are inappropriate for the true state of the spacecraft. This increase in risk though the possibility of exacerbating a failure through continued operation rather than safing is one of the items we are asking missions to trade against the small expected estimated in the previous section in order to justify MBD. For the Livingstone flight experiments, risk was acceptable because the MBD system operated on top of a complete, separate fault protection system which protected the spacecraft. In addition, Livingstone’s communication with the spacecraft was through a filter which ensured Livingstone could only send specific commands which had been analyzed for safety. If the underlying fault protection system were activated, that filter would be closed and Livingstone would be terminated.

In an unforgiving environment such as space, Livingstone’s ability to provide novel diagnoses and recoveries to failure combinations we had not explicitly considered was far less important than being able to verify exactly how it was going to respond in the most likely and most critical anomaly situations. Guarding an MBD system with a traditional fault protection system and restricting the commands it can give is one approach to bringing it to the level of predictability needed to convince mission stakeholders the spacecraft will not go in to an unsafe state. This strategy was appropriate for experiments whose purpose was to show the technology could be flown. In routine operations, it would tend to undermine the cost and value arguments of using MBD.

10 Discussion

In this paper we have discussed the expectations for model-based diagnosis and recovery systems such as Livingstone and why we believe not all of those expectations were met. We attempted to lay out the basic cost/benefit drivers in a domain of interest (unmanned spacecraft) and our understanding of why model-based diagnosis and recovery have found relatively little traction. We can grossly characterize the common practice in fault protection to be identifying those contingencies where an active, specific response to anomalies must be made (e.g., loss of a motor during an orbital insertion) and providing identification and response to those states. In other anomalous conditions, the spacecraft is safed and engineers diagnose the problem post hoc. For the missions we’ve considered, this approach seems to provide lower risk and more than adequate value in terms of anomaly response when compared to MBD. In addition we have not yet developed or seen an argument or demonstration that the total analysis, development and testing cost for the common practice is higher than an alternative based upon MBD.

During non-critical mission phases, the net value of having on-board diagnosis and recovery is low since we are free to simply put the spacecraft into a safe mode and only then invest resources attempting to find a diagnosis or response. During critical phases (as well as non-critical) the real need to circumscribe and validate the responses of the fault protection system decreases the proposed value of MBD’s ability to generate novel responses, while increasing its testing and analysis costs in an attempt to contain risk. We also believe the key questions of how on-board, component-level diagnosis fits in with and adds value to the broader task of fault protection engineering, and how MBD technologies would reduce fault protection costs remain open. Thus we believe it is difficult to justify the use of on-board, generative diagnosis and recovery systems like those we have been involved with based on cost, risk or value, at least for the type of missions with which we are familiar.

11 Related Work

Researchers at Xerox PARC developed a model-based diagnosis system for copiers [Bell et al.1991] intended to assist field technicians. After being presented to technicians, the system was not deployed and a community knowledge sharing system was deployed instead [Bobrow and Whalen2002]. To paraphrase, the model-based diagnoser was not deployed because technicians knew how to identify and correct common faults, and small optimizations in that process were not of high value. The real issue was unexpected issues that were not foreseen during the design of the machine or development of the diagnostic models. Researchers have applied real-time model-based reasoning to a UH-60 Helicopter [Patterson-Hine et al.2001] using the TEAMS and TEAMS-RT systems [Deb et al.1995]. Users model signal propagation between system components and placement of sensors. TEAMS performs off-line testability analysis to determine which component failures can be detected and further isolated, and make recommendations to improve testability. TEAMS has been used for testability analysis of several large aerospace systems. TEAMS-RT uses the same model to determine which components may be failed from pass/fail outcome of sensor tests. We were unable to find references indicating whether TEAMS-RT has been used in an FP system or ground-based diagnostic aid during aerospace operations. The very successful Cassini mission, whose main propulsion system was later used as a benchmark problem in development of Livingstone, made use of rule-based fault diagnosis and recovery system in operations [Hackney et al.1993, Brown et al.1995]. Through a FMECA process, the set of critical failures, the symptoms that would result, and the appropriate responses were derived. Mappings from monitored values to diagnosed states were encoded in rules. Commands to respond to each state were similarly encoded.
References


Prioritising Model-Based Debugging Diagnostic Reports

Wolfgang Mayer† and Rui Abreu‡ and Markus Stumptner† and Arjan J.C. van Gemund‡

†Advanced Computing Research Centre
University of South Australia
Australia
{mayer,mst}@cs.unisa.edu.au

‡Embedded Software Lab
Delft University of Technology
The Netherlands
{r.f.abreu,a.j.c.vangemund}@tudelft.nl

Abstract

Model-based debugging has proved successful as a tool to guide automated debugging efforts, but the technique may suffer from large result sets in practice, since no means to rank or discriminate between the returned candidate explanations are available. We present a unique combination of model- and spectrum-based fault localisation approach to rank explanations and show that the combined framework outperforms the individual approaches as well as other state of the art automated debugging mechanisms.

1 Introduction

The problem of faulty software has been recognised as long-standing issue, with considerable costs attached to locating and eliminating problems in development as well as after deployment of software systems [19]. In particular, testing, validation and debugging of software consumes a considerable slice of the overall software development costs. Hence, numerous approaches have been proposed to automate parts of this process to help detect more defects earlier in the development cycle and to guide software engineers towards possible faults.

Early debugging efforts were geared towards reducing the size of a program that must be investigated by analysing the structure and dependencies between different parts of the program’s source code [16]. More recently, dynamic analysis techniques have been proposed that exploit traces of program executions to accommodate the size of modern software systems [23]. While applicable to a wide variety of programs, both approaches are limited by the absence of a detailed model of the correct behaviour of a program.

To overcome these limitations, a spectrum of model-based fault isolation techniques has been advocated as powerful debugging aid that can help to isolate faults in complex programs [12]. By comparing the state and behaviour of a program to what is anticipated by its programmer, model-based reasoning techniques separate those parts of a program that may contain a fault from those that cannot be responsible for observed symptoms. A distinguished advantage of the model-based framework is that it helps programmers by systematically exploring different fault assumptions while hiding the complex underlying reasoning procedures behind a simple intuitive conceptual interface.

Initial experiences with model-based software debugging (MBSD) have shown that the approach is competitive with other state of the art automated debugging aids [13], but it has also become clear that no single technique is sufficient to deal with a variety of programs and faults. Instead, a combination of approaches must be pursued where the strengths of individual techniques complement each other to lead to a more accurate and robust debugging tool.

In this paper we present a combined framework that integrates model-based debugging with popular dynamic program analysis techniques to focus search and rank results. We show that, as a result, fewer program fragments are being implicated, leading to considerably increased accuracy as well as reduced computational complexity of the overall approach. While MBSD is general enough to be combined with almost any debugging tool that can expose its findings in terms of the original program’s source code and a set of fault assumptions, the combination of semantic and trace-based analysis is particularly appealing, since the approaches contribute complementary information: MBSD injects and analyses specific modifications to the semantics of a program, while dynamic analysis exploits fault correlation to focus the search.

Our presentation is structured as follows: The principles of model-based debugging are outlined in Section 2, followed by a discussion of spectrum-based fault localisation in Section 3. The combined framework is discussed in Section 4. Empirical validation of our approach and our findings are given in Section 5. Section 6 discusses relevant related works, followed by the conclusion.

2 Model-based Software Fault Localisation

In search for effective (semi-) automated debugging aids, many different strategies have been proposed in the last three decades. Approaches to automate analysis and isolation of faults in programs range from purely syntactical checks to isolate common fault patterns [4], over execution trace-based analysis [15] to full-fledged semi-automatic program verification [2]. Syntax-based analysis can be easily applied to most programs, but its results are often language-specific and depend on particular syntactic programming styles; trace-based techniques depend on a suitable test harness being available.

Better results can often be achieved if a model of the correct program behaviour is available to guide debugging efforts. For example, a partial specification expressed in some formal lan-
guage. Unfortunately, building such models is error prone and prohibitively expensive for many software development scenarios. Attempts to devise formal specifications for non-trivial systems has shown that constructing a model that captures an abstraction of the semantics of a system can be as difficult and error-prone as building a concrete implementation [14].

Model-based debugging [12] aims to close the gap between powerful formal analysis techniques and execution-based strategies in a way that does not require the end-user to possess knowledge of the underlying reasoning mechanisms. Here, an adaption of the classic “reasoning from first principles” paradigm borrowed from diagnosis of physical systems is particularly appealing, since much of the complexity of the formal underpinnings of program analysis can be hidden behind an interface that resembles the end-user’s traditional view of software development.1

Different to classical model-based diagnosis, where a correct model is furnished and compared to symptoms exhibited by an actual faulty physical artifact, debugging software reverses the roles of model and observations. Instead of relying on the user to formally specify the desired program behaviour, the (faulty) program is taken as its own model and is compared to examples representing correct and incorrect executions. Hence, the model in MBSD reflects the faults present in the program, while the observations indicate program inputs and correct and incorrect aspects of a program’s execution. Observations can either be introduced interactively, or be sourced from existing test suites.

Example 1 Consider the program in Figure 1. An observation for this program consists of concrete program inputs, that is values for variables tbl, n and k before line 1, together with the anticipated result value returned by the algorithm. For example, the assignments $tbl \leftarrow [90, 21, 15, 0, 0, 0, 8, 23, 0, 0, 0, 0, 0, 0, 0, 0, 50, 60, 59]$, $n \leftarrow 16$, $k \leftarrow 90$ and the assertion result $= 0$ could be an “observation” specifying the inputs and the desired result of a particular program execution.

Since the result $(-1)$ obtained by running the program on the given inputs contradicts the anticipated result (0), it has been shown that the program is incorrect. (Indeed, the program contains a defect in line 9: when assigning 0 to variable i, the program works as expected.)

In the following, we briefly outline the model construction. More detailed discussion can be found in [11; 12]. A program is partitioned into “components”, each representing a particular fragment in the program’s source code. The behaviour of each component is automatically derived from the effects of individual expressions the component comprises. Connections between components are based on control- and data-dependencies between the program fragments represented by each component.

Example 2 Assume a model at statement granularity is to be created from the program in Figure 1. For each statement $s$, a separate component is created that is comprised of the expressions and sub-expressions in $s$. The inputs and outputs of the components correspond to the used and modified variables, respectively. Connections between the components are created to reflect data dependencies between statements in the program (as determined by a simple data flow analysis). Additional variables and components may be introduced to correctly capture data flow at points where control flow paths may split or merge.

The component $C_7$ corresponding to statement 7 in Figure 1 is represented as a component with input $i_2$ and output $i_7$. Here, $i_7$ represents the result value of statement 7, and $i_2$ denotes the previous value of variable $i$ that is implicitly defined at the loop head in line 2.

Similar to classical model-based diagnosis, the model also provides different operating modes for each component, where the “correct” mode $\neg AB(C)$ of component $C$ corresponds to the case where $C$ is not to blame for a program’s misbehaviour. In this case, $C$ is defined to function as specified in the program. Conversely, when $C$ is assumed “abnormal” ($AB(C)$), the $C$ may deviate from the program’s behaviour.

Example 3 The behaviour of $C_7$ can be expressed as the logical sentence

$$\neg AB(C_7) \Rightarrow i_7 = i_2 + 1. \quad (1)$$

In the case where $C_7$ is considered faulty ($AB(C_7)$ is true), the effect on $i_7$ is left unspecified.

The main difference between the original program and its model is that the model represents the program in a form that is suitable for automated consistency checking and prediction of values in program states in the presence of fault assumptions. This includes program simulation on partially defined program states and backward propagation of values or constraints, which would not occur in a regular (forward) program execution.

Since the resulting model includes the same faults as the program, means to compensate for incorrect structure and behaviour of components must be introduced. While heuristics to diagnose structural deficiencies in physical systems can be

---

1In the software engineering context, the notion of “first principles” may be interpreted in the sense of “directly available from program execution and source code”.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.

Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
based on invariants and spatial proximity [3], in software, the model must be adapted and restructured once a defect in its structure has become a likely explanation. Here, detection and model adaption must be guided by using abstract assertions that capture simple “structural invariants” [11]. Also, since different fault assumptions may alter the control and data flow in a program, models may be created on-the-fly rather than in the initial setup stage.

A trade-off between computational complexity and accuracy can be achieved by selecting different abstractions and models [12], both in terms of model granularity and representation of program states and executed transitions.

**Example 4** In Example 3 the representation of program state has been left unspecified. Using an interval abstraction to approximate a set of values, sentence (1) becomes a constraint over interval-valued variables $i_2$ and $i_7$ [13]. Another possible abstraction is to encode the operation as logical sentences over the variables’ bit representations [13]. In this paper, we use the interval abstraction described in [12], since it provides good accuracy but avoids the computational complexity of the bit-wise representation.

Similar to consistency-based diagnosis in physical systems [17], from discrepancies between the behaviour predicted by the model and the behaviour anticipated by the user, sets of fault assumptions are isolated that render the model consistent with the observations. Diagnoses are obtained by mapping the implicated components into the program’s source code. Formally, our framework is based on extensions to Reiter’s consistency-based framework [12].

**Definition 1 (Diagnosis)** Let $P$ denote a program and $T$ a set of test cases, where each $T \in T$ is a pair $(I, A)$ where $I$ specifies $P$’s inputs and $A$ is a set of assertions over variables in $P$ that (partially) specify the correct behaviour of $P$ with respect to $T$. Let $C$ denote a partition of the statements in $P$.

A diagnosis of $P$ with respect to $T$ is a set of components $D$ such that $\forall (I, A) \in T$:

$$P(I) \land \{AB(C) \mid C \in D\} \land \{\neg AB(C) \mid C \in C \setminus D\} \neq \neg A.$$ 

**Example 5** Continuing Example 4, a contradiction between the test case from Example 1 and the program is detected when the assertion checking the expected result fails. It is derived that the (cardinality-)minimal fault assumptions that are consistent with our test specification are: $\{AB(C_1)\}$, $\{AB(C_2)\}$, $\{AB(C_3)\}$, and $\{AB(C_12)\}$. Hence, the statements in lines 1, 7, 9 and 12 are considered the possible root causes of the symptoms. Any other statement cannot alone explain the incorrect result, since the result remains incorrect even if a statement is altered.

**2.1 Issues in MBSD**

While the pure MBSD framework is well-suited to carry out complex inferences, its application in practice is limited due to the following factors:

**Result interpretation:** If many explanations are returned, MBSD alone provides little information to discriminate between the different explanations. Here, a mechanism to rank results would be desired.

Different to electronic circuits, where long sequences of e.g. inverters are uncommon, program executions frequently contain long chains of control- and data dependencies, leading to a number of explanations that cannot be distinguished without further observations. For example, the value of the conditional test in line 2 depends on all statements executed in previous iterations. Interactive measurement selection techniques are difficult to apply, since program states in different executions may be incomparable, rendering entropy-based solutions ineffective. Returning a “super component” as an explanation is also not viable in general, since the involved statements can span many different program fragments. Therefore, an approach that works with little or no user involvement is desired.

**Scalability:** The application of MBSD has been limited to small programs, since the computational effort exceeds what is considered reasonable for interactive scenarios. Hence, inference processes must be applied selectively to remain efficient.

**External interfaces:** MBSD requires that effects of program fragments can be simulated even if only partial information is available. Programs interacting with external components, such as I/O, files and GUIs, must be modified to either remove these interactions or provide placeholder implementations.

The first two issues can be addressed by introducing a mechanism to estimate, for each component $C$ in the model, how likely it is that $C$ contains a fault. The third issue is common to most program analysis techniques and is beyond the scope of this paper.

Assuming a suitable measure is available, ranking of results based on fault probability and investigating different explanations in best-first order rather than computing all explanations at once are straightforward. Since a priori probabilities are typically not directly available, other means to determine a suitable likelihood value must be used.

The aim of this paper is to show that correlation between the execution patterns of statements with correct and failed executions can significantly improve diagnosis results. The following section outlines our approach to assessing the similarity between different program executions and test outcomes. Since MBSD does not usually exploit correct program executions in any way, this approach can contribute valuable information to guide the model-based framework.

**3 Spectrum-based Fault Localisation**

When more than one test case are available, dynamic program analysis techniques have shown that comparing the program behaviour over multiple test runs can indicate which program components may be likely to contribute to an observed symptom.

In the following, we assume that a program $P$ comprises a set of $M = |C|$ components (statements in the context of this paper) and is executed using $N = |T|$ test cases that either pass or fail.

Program (component) activity is recorded in terms of program spectra [$1; 8$]. This data is collected at run-time, and
M components error vector

\[
\begin{bmatrix}
o_{11} & o_{12} & \ldots & o_{1M} & e_1 \\
o_{21} & o_{22} & \ldots & o_{2M} & e_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
o_{N1} & o_{N2} & \ldots & o_{NM} & e_N
\end{bmatrix}
\]

Figure 2: Participation Matrix O

N spectra

Each component the Ochiai similarity \( s_j \) is given below the matrix. For \( C_3 \), the similarity coefficient \( s_3 \) is 0.63: as can be seen from the third column in the matrix, there are two failing test runs where \( C_3 \) is executed \( (a_{31}(3) = 2) \), no failing run where \( C_3 \) does not participate \( (a_{10}(3) = 0) \), and three successful executions where \( C_3 \) is involved \( (a_{10}(3) = 3) \).

\( C_6 \), \( C_2 \) and \( C_9 \) are considered to be most closely correlated with failing tests and should be examined first. Conversely, \( C_4 \) is not considered relevant at all.

Recent studies on spectra-based fault localisation indicated that this scheme is effective even for small test suites containing only few test cases [1]. For the programs investigated here, good fault localisation was achieved when using six failing test cases and twenty passing runs.

4 Spectra-Enhanced MBSD

As SFL functions without a semantic model of the program, the technique is easily applied. Experiments with different similarity measures have shown that Ochiai similarity generally outperforms other spectra-based indicators and can give good hints on the location of a fault in a program [1].

At the same time, the absence of a model also limits the accuracy of fault localisation. Even for comprehensive test suites, the execution patterns of some components may not be distinguishable, and faulty components may show erroneous behaviour only in particular execution contexts. As a result, the similarity measure may implicate unrelated program fragments.

Conversely, the model-based technique captures the semantics of programming constructs, but does not assign ranking information to candidate explanations. Furthermore, model-based diagnosis traditionally only considers discrepancies, but does not utilise correct test cases, although all failing test cases will be considered. In contrast, spectra-based methods exploit both correct and failing test runs to rank candidates. Hence, both techniques complement each other.

Algorithm 1 outlines our combined approach. The algorithm executes in three stages, with the similarity-based approach used in the setup stage (steps 1 to 5), feeding into the subsequent model-based filtering stage (steps 6 to 16), followed by an optional best-first search stage (lines 17 to 24). This combination has significantly lower resource requirements than applying MBSD on the whole program but using SFL only to rank results. We start by partitioning the program \( P \) into a set of components \( C \) and execute \( P \) on the available test cases \( T \) to obtain the participation matrix \( M \). Using \( M \), we partition \( T \) into passing tests (\( T_F \)) and failing ones (\( T_F \)). From \( M \), the Ochiai similarity vector is computed; its values are subsequently assigned to components as a-priori fault probabilities to yield the component list \( \hat{C} \) sorted by fault probability.

In the subsequent loop, candidate explanations are computed using the MBSD approach to isolate the most likely explanations based on \( \hat{C} \) and \( T_F \). While it is possible to apply MBSD once to compute all explanations and present the ranked candidates to the user, an incremental strategy permits

\[2\] We use the term probability as synonym for likelihood to be incorrect. Our measure does not necessarily conform to the laws of probability theory.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Algorithm 1 Spectra-Enhanced MBSD Algorithm

Inputs: Program $P$, set of test cases $T$

Output: Fault assumptions explaining failed test runs

1. $C ←\text{CREATECOMPONENTS}(P)$
2. $M ←\text{GETCOMPONENTMATRIX}(C, T)$
3. $(T_P, T_F) ← T$
4. $S ←\text{COMPUTESIMILARITY}(M)$
5. $\tilde{C} ←\text{ASSIGNCOMPONENTPROBABILITIES}(C, S)$
6. $t ← 1 + \epsilon$
7. $R ← \tilde{C}$
8. repeat
9. $D ← MBSD(\tilde{C}, T_F, t)$
10. if $D_{\text{bug}} \in D$ is confirmed then
11. return $D_{\text{bug}}$
12. else
13. $t ←\text{PROBABILITY}(D)$ for some $D ∈ D$
14. end if
15. $R ← R \setminus \bigcup_{D ∈ D} \text{COMP}(D)$
16. until $D = ∅$ or $t = 0$
17. while $R \neq ∅$ do
18. $C ← \arg\max_{C ∈ R \land \exists C'\in\text{NEIGHBOURS}(C), C' ∉ R} \text{PROBABILITY}(C)$
19. if $C$ is confirmed faulty then
20. return $\{AB(C)\}$ \hspace{1em} $\triangleright$ Partial explanation
21. end if
22. $R ← R \cup \{\text{NEIGHBOURS}(C) \cap R\} \setminus \{C\}$
23. end while
24. return No explanation found

the algorithm to stop early once a fault has been identified. In each iteration, the user is presented a number of candidate explanations for examination. If the actual fault has been located, the algorithm stops; otherwise, none of the candidates represent valid explanations and other candidates must be generated. The algorithm stops once no more explanations could be found or if none of the remaining components was executed for a failing test.

We modified the basic MBSD algorithm to return only the explanations with probability $p$ less than a given threshold $t$, with the additional restriction that $p$ is maximal among the returned diagnoses. Hence, only explanations with the same likelihood are returned in an iteration of Algorithm 1. Initially, $t$ is set to a value slightly larger than 1 (the Ochiai similarity is always $\leq 1$), hence candidate explanations with maximum likelihood are enumerated first. By decreasing $t$ in each iteration, lower-scoring alternatives are explored if no higher-scoring candidate has been confirmed by the user. Our implementation of MBSD caches intermediate models and conflicts to avoid repeated computations.

If no explanation is found after all components implicated by MBSD have been explored, we employ a best-first search procedure that traverses the program along dependencies between components with decreasing fault probability. No explanation may be found if a fault in the program has larger cardinality than the MBSD threshold or if the fault affects component inter-dependencies such that the fault assumptions and model abstraction can no longer represent the fault. Function NEIGHBOURS($C$) returns the set of components that are directly connected to $C$ in $P$, COMP($D$) returns the set of components that occur in diagnosis $D$, and PROBABILITY($C$) returns the fault probability assigned to component $C$ by SFL. In line 18, the component with maximum fault probability that is connected to a previously explored component is selected. If the component is confirmed to be (part of) a valid explanation, the search stops and the diagnosis is returned. Note that the explanation may only cover part of the true fault. Line 24 in Algorithm 1 can only be reached if the faulty program fragment is not covered by any component, or if the user oracle that decides whether an explanation is indeed a satisfying explanation is imperfect and may miss a fault.

We employ the common assumptions that components may fail independently. While faults in a statement can imply failure in subsequent statements due to data dependencies, this need not be true in general; since most faults in our test suite are confined to a single component and only few statements occur where implied faults are possible, this assumption has not significantly affected the outcomes of our study. Also, since fault probabilities are estimated from correlation with failing tests, different components participating in the same failure will be assigned higher similarity, partially compensating for missed component fault interactions.

Example 7 Applying Algorithm 1 using the test suite from Example 6, $\{AB(C_7)\}$ and $\{AB(C_9)\}$ form the set of candidate explanations. Both candidates are associated with the highest similarity coefficient 0.71.

Notably, this result improves upon both individual fault localisation procedures. Different from pure SFL, $\{AB(C_6)\}$ is no longer considered an explanation. Conversely, candidates $\{AB(C_1)\}$ and $\{AB(C_2)\}$ obtained using pure MBSD are low-ranking in SFL and hence omitted at this stage. $\{AB(C_12)\}$ is already eliminated by pure MBSD when using the second failing test case introduced in Example 6.

Without further information, neither approach can discriminate between the two remaining candidate explanations. Since it is assumed that the user acts as oracle that can reliably recognise true faults, the algorithm stops after the first iteration, once the statement in Figure 1 corresponding to $\{AB(C_9)\}$ has been confirmed to be incorrect.

Otherwise, the diagnosis threshold $t$ would be set to 0.71 and the algorithm would continue to present $\{AB(C_1)\}$ as the (last) remaining alternative explanation.

The use of similarity measures to guide diagnosis can potentially lead to considerable savings; moreover, the behaviour of Algorithm 1 degrades gracefully if components with high probability do not actually correspond to faults. In the worst case, the number of diagnoses to be examined by the user is the same as when using the non-guided MBSD strategy. In the next section we evaluate our algorithm on a larger test suite.

5 Empirical Evaluation

To gain a better understanding of the combined approach, the TCAS program was taken from the Siemens Test Suite\(^3\), a test suite...
bench commonly used in the debugging community. The program simulates the resolution-advisory component of a collision avoidance system similar to those found in commercial aircraft. The program consists of 138 lines of C code and takes twelve parameters as input; the numeric result value encodes one out of three possible resolution advisories. The program is equipped with 1608 test cases and 41 different variants with known faults. For each variant, on average, forty test cases reveal a fault. In our experiments, all available test cases were used.

As the efficiency of debugging in practice often depends on the experience of the software developer, comparing different approaches is difficult. In particular, simple precision and recall-based evaluation may be insufficient, since the structure of a program is not taken into account. In an attempt to devise an objective measure to assess the quality of automated debugging aids, a quality metric that exploits dependencies within a program has been proposed, where the quality of a report depends on the fraction of a program that need not be examined given the debugging tool’s output. Starting with the program elements implicated by an automated debugger, dependencies between program elements are traversed in breadth-first order until the fault has been reached. This strategy aims to mimic programmer behaviour, where possible influences along control- and data flow paths are explored. The fraction of the program that has not been traversed leads to the quality indicator [18].

In our framework, if the explanation covering the true fault is ranked \( n^{th} \) among the candidate explanations, the fraction of the program that is traversed is given by

\[
q = \frac{\sum_{i=1}^{n} \text{STMNTS}(D_i)}{|P|},
\]

where \( D_i \) represents the \( i^{th} \)-ranked candidate explanation, \( \text{STMNTS} \) is a function that returns the set of program statements covered by the components in a diagnosis, and \(|P|\) denotes the number of statements in the program. Otherwise, if Algorithm 1 stops without locating the true fault, the breadth-first search procedure is invoked starting with all implicated statements

\[
\bigcup_{D \in \text{MBSD}(C, T_{F, 1+\epsilon})} \text{STMNTS}(D)
\]

to obtain \( q \) as outlined previously.

5.1 Experimental Results

Results for the individual approaches have already been published elsewhere; an evaluation of MBSD is presented in [13], where an interval abstraction is applied to yield, on average, nine statements to be inspected. The median quality indicator is 0.87. The results obtained with SFL are discussed in [1]. Exploring the program beginning with the highest-ranked statement, the true fault is located after 28 statements have been examined on average. The resulting median report quality is 0.86. Figures 3 and 4 summarise the overall outcomes.

To obtain a first impression how well the two approaches complement each other, Figure 3 contrasts the components implicated by either model with those blamed by both. It can be seen that the AIM significantly reduces the number of components, and that neither model subsumes the other.

Restricting the debugging process to those statements that appear in both models, the median number of statements reduces from 36 to 8. Hence, the total number of statements considered relevant reduces considerably when using the combined approach.

Similar improvements can be observed in the ranking of components. Using the pure SFL approach one hits the true fault after inspecting twenty statements on average, but many unrelated statements must be examined. When using MBSD as filtering mechanism, the true fault is located after seven statements on average. Hence, the model-based filtering mechanism seems well-suited to prune away irrelevant components from the SFL fault profiles.

The improved accuracy of the combined approach also reflects in much improved quality indicators. Figure 4 depicts the quality measure obtained for the individual 41 test programs using our fault localisation approaches. It is observed that the combined approach largely outperforms the individual techniques. In some cases, SFL outperforms the combined approach, suggesting that the model used in MBSD may not be able to accurately reflect the fault. This difference may also be attributed to the execution of some faulty code correlating well with the failing test cases. If the MBSD part of our algorithm cannot precisely locate the fault, the SFL method can score higher. So far, we have not been able to devise heuristics that can consistently predict this discrepancy from the a-priori component probabilities and diagnoses to further improve accuracy. Since both approaches use heuristics to rank candidates, it may be the case that one method outperforms the combination on individual candidates, even though it is superior overall.

Overall, the fraction of the program that must be inspected reduces from 13% and 30%, respectively, to 8%. Although MBSD alone is not able to locate faults for 9 of the 41 programs (due to limitations on faults in global variable initialisation in our current implementation), the overall performance of the combined approach does not seem to be adversely affected in most cases. This can be explained by two observations: (i) the number of diagnoses that are implicated in those cases is small (4 on average), and (ii) the suspect program fragments are close to the actual faults when navigating the program structure.
We also evaluated a modified version of Algorithm 1, where the MBSD section is stopped after the six\footnote{This cutoff seemed to have the best overall effect for an extended test suite used in [1].} most highly ranked components have been explored; the remaining components were subsequently explored using the best-first part of our algorithm. The resulting quality indicators are labelled Top\textit{6} in Figure 4. The results indicate that the components implicated by the combined approach sometimes narrowly miss the true faults; in these cases, the score measure improves compared to the combined approach. In other cases, following the original algorithm is more successful. Overall, the quality indicators do not differ significantly between the two models. Investigating whether heuristics can be developed that choose a cutoff to improve accuracy remains for future work.

Figure 5 visualises the number of located bugs for different fractions of inspected code. Our approach vastly outperforms the simple spectrum-based fault localisation techniques proposed in [18], where different combinations of union and intersection of “similar” passing and failing test runs are computed. This can be attributed to the improved ranking mechanisms built into our algorithm that is more robust with respect to overlapping passing and failing executions. Our combined approach also improves with respect to SOBER [10], a statistical approach based on hypothesis testing that has been shown to dominate other recent bug detectors.

Delta slicing and \texttt{explain} [7] are two techniques for fault localisation that exploit differences between passing and failing abstract program executions traces found by a model checker. Comparing our results to the published results in [7], we conclude that the combination of SFL and MBSD is far superior than \texttt{explain} (which requires the user to explore 24–64\% of a program) and performs competitive with respect to Delta Slicing (within 5\%). Interestingly, when using the cutoff variant of our algorithm described in Figure 4, our approach also dominates Delta Slicing. (This comparison is not exhaustive, since results for only a small subset of the examples considered in our study is available for the competing approaches.)

6 Related Work

Several systems employing dynamic analysis techniques for fault localisation are present in the literature. Tarantula [8] obtains program spectra from test case executions and graphically visualises the fault proneness indicator based on participation of individual statements in passing and failing runs. Tarantula does not exploit information about the anticipated behaviour of a program and hence relies on external tools to assess the outcome of test runs.

Machine learning techniques have been applied to programs [21] and their executions [15] to infer likely invariants that must hold at particular locations in a program. Violations can subsequently be used to detect potential errors. Model-based approaches have been shown to provide more reliable behaviour than [15], since success of the trace analysis depends much on the test runs and type of invariants to be inferred [9]. The static program analysis approach requires that similar patterns appear repeatedly in a program, but is not applicable when common patterns are not easily identified.

Combining program execution and symbolic evaluation has been proposed to infer possible errors [5]. Similar to MBSD, a symbolic, under-constrained representation of a program execution and memory structures are built. Instead of using fault probabilities to guide diagnosis, only those candidate explanations that definitively imply a test failure are flagged. Hence, the tool complements our approach by highlighting a subset of all provable faults in a program, while our approach aims at identifying those program fragments that may contribute to a fault.

Model-based debugging has been explored using a variety of different abstractions of concrete programs [12]. Recently, similar techniques have also been proposed to isolate specific faults stemming from incorrect implementation of high-level conceptual models [22]. Mutations applied to state machine models allow to detect conceptual errors, such as incorrect control flow and missing or additional features found in the implementation compared to its specification. Model-based test generation [6] from abstract specifications of systems...
employs a similar idea where possible faults manifested as differences in abstract state machines are analysed to generate tests. Our work differs in that we are concerned with program representations that more closely reflect the actual program artefact to locate faults at a more detailed level. While initial steps to integrate similar conceptual abstract models have been undertaken in an attempt to isolate “structural” faults [11], detailed analysis remains future work.

Diagnosis and repair in the context of distributed systems composed from Web Services has also been investigated [20]. In particular, diagnosability and analysis of diagnosis and repair plans are central parts of this work. Similar ideas are in principle applicable in the debugging context, but further work is required to devise a suitable analysis framework that can operate on the more implementation-centric view employed in this work.

7 Conclusions & Future Work

We have shown that the accuracy of model-based debugging increases significantly when applied in combination with complementary approaches that estimate fault probabilities. The unique combination of semantics-based analysis as undertaken in MBSD and dynamic aspects obtained from program execution spectra has proved to focus debugging efforts; overall, a reduction of user effort to less than 10% compared to the complete program has been achieved on our test suite. We have further shown that our approach is among the state of the art automated debugging tools.

Several issues for further research remain: On the MBSD side, connecting the lower-level models that reflect the program to high-level conceptual models to detect a more diverse set of faults seems promising to address current limitations. The same idea may be useful to aid model selection and focus fault assumptions. On the dynamic analysis side, introducing machine learning techniques to infer likely invariants that can then be used to further filter and guide the MBSD modelling efforts are possible avenues worth further exploration.

References


How to debug Sequential Code by means of Constraint Representation

Mihai Nica and 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
{mihai.nica,jweber,wotawa}@ist.tugraz.at

Abstract

In this paper we discuss a model-based diagnosis approach to the debugging of sequential programs which is based on expressing the program semantics by a constraint representation. We formalize this representation and prove its equivalence with respect to the semantics of the programming language, and we frame the debugging process as a constraint satisfaction problem (CSP). We identify two main advantages of a constraint representation of sequential programs. First, the constraint representation allows for a straightforward integration of program annotations like pre- and post-conditions and invariants. We show that the integration of those additional specifications can improve the diagnostic precision by eliminating diagnoses. Second, we can benefit from the extensive research work which has already been done on CSPs. In particular, we propose to derive a metrics for the complexity of debugging from the structural decomposition properties obtained for the corresponding CSP. We classify the complexity of the diagnosis problem by using the hypertree width of the hypergraph for the CSP, and we report experimental results concerning the hypertree width for a set of programs.

1 Introduction

Debugging, i.e., the detection, localization, and correction of bugs, is an important task in software engineering. A lot of research has already been devoted to the automation of software debugging, but most of this work has concentrated on fault detection only. For example, model-checking and other verification techniques are gaining increasing interest, not only in the research community but also in industry. However, there is still a lack of work on automated fault localization and correction techniques.

Artificial Intelligence offers techniques like model-based diagnosis which can be effectively applied to this field. Friedrich et al. [5], Stumptner and Wotawa [14], Mateis et al. [11], Mayer et al. [12], and most recently Köb and Wotawa [10] are examples for the application of model-based diagnosis to program debugging. There are also a lot of papers dealing with debugging which originate from other fields. For example, Staber et al. [13] and Griesmayer et al. [8] incorporate model-checking techniques into debugging. These techniques use some ideas from model-based diagnosis and allow for the integration of debugging and verification to some extent. Many of these approaches rely on test cases defining the correct (expected) input/output-behavior of the program.

The past works on the application of model-based diagnosis to program debugging have investigated different modeling approaches, in particular dependency-based models (e.g., [5]) and value-based models (e.g., [12]). However, there is another possibility of modeling a program which has so far gained little attention: the program semantics can also be expressed by constraints, and debugging can be viewed as solving a constraint satisfaction problem (CSP) for a given test case.

This approach is very promising due to several reasons. First, it allows for a full integration of program annotations like pre- and post-conditions and invariants. The enrichment of program code with such annotations has become very popular among software engineers in the past few years, supported by many tools which have recently been developed, e.g., for the Java programming language. In the context of dependency-based models, the authors of [23] proposed to employ annotations for the model-based diagnosis of programs, but they only utilize the dependencies between variables indicated by the annotations rather than the complete semantics of the specification. The full integration of the specifications provided by annotations can significantly improve the diagnostic precision without imposing any significant overhead.

Another advantage of using constraint representations is the fact that we can benefit from the large amount of existing research work on CSPs. Apart from the possibility of using approved CSP algorithms, this also allows us to rely on theoretical results like, e.g., insights concerning the complexity of solving CSPs.

The authors of [24] proposed to diagnose errors in programs using constraint programming. Their approach requires that the programmer provides contracts, i.e., pre- and
post-conditions, for every function. However, the authors do not investigate the complexity of solving the resulting problem and the scalability to larger programs. In particular, they do not consider structural decomposition or other methods which could make the approach feasible.

Our novel contribution is threefold. First, we define a programming language comprising constructs similar to those in commonly used sequential languages, and we formally describe how a program written in this language can be converted to a constraint representation. We also prove that the intermediate representations and the resulting CSP are semantically equivalent to the original program. Second, we explain the integration of program annotations in our constraint representation, and we show by means of an example that annotations can improve the diagnosis results by ruling out diagnoses which otherwise would be possible. Our approach can utilize annotations at any location in the code, and it does not impose any requirements on the completeness of such specifications.

Third, we investigate the complexity of solving the resulting CSP. In the past, much work has been done on the structural decomposition of CSPs in order to find problem classes which are tractable. Gottlob et al. proposed the hypertree decomposition, and they showed that this decomposition method generalizes other important methods [6; 7]. The hypertree width, a characteristic of the structure of the constraint system, is a measure for the complexity of solving a CSP. Hence, we propose to classify debugging problems by the hypertree width of the hypergraph corresponding to a specific program. In other words, by performing a hypertree decomposition we can obtain a metrics for the complexity of debugging a certain program, even without performing the actual debugging, i.e., solving the corresponding CSP.

We present first empirical results on the hypertree width for a set of different programs. In particular, our experiments focus on the question whether the hypertree width of general programs comprising loops has an upper bound. If this is the case, then debugging such a program would be - from a theoretical point of view - tractable. Our results indicate that this may be the case; on the other hand, they show that even for small programs the hypertree width may be very large, which means that the debugging process is very complex.

This paper is organized as follows. The following section describes the transformation of sequential programs to CSPs. Section 3 shows by means of an example that utilizing annotations can improve the diagnosis, and we outline the integration of annotations in the constraint representation. Section 4 gives a background on the hypertree decomposition and the hypertree width. Then we provide an overview of the entire debugging process and explain the applied algorithms. In Section 5 we present experimental results mainly concerning the hypertree width of a set of programs. Finally we discuss related research and conclude the paper.

2 Representing Sequential Programs as CSPs

The conversion of sequential programs to CSPs involves several issues:

1. We define a general programming language which can be converted by our approach.

2. As loops cannot be directly represented as a constraint system, we have to transform the original program to a loop-free variant which is equivalent.

3. The constraint representation requires that the left-side variables in the program have unique names. Moreover, conditional statements cannot be directly translated to constraints. Hence, we rely on an intermediate representation, the Static Single Assignment (SSA) form, which resolves these problems.

4. The SSA form can now be converted to a constraint representation.

2.1 The language J

We define an assignment language J with a syntax and semantics similar to well-known languages like Java, but without object-oriented constructs. For simplicity J does not support procedure calls, but it should be noted that J can be straightforwardly extended for integrating procedures.

J comprises assignments, conditionals, and loop statements. It is defined over a datatype D, where D comprises a domain A_D, a set of functions F_D, and a set of predicates P_D. We further assume that there is a one-to-one correspondence between the domain, the functions, and predicates on the one side and their syntactical representation in J on the other side. Note that in the following we do not distinguish between the syntactical representation of datatypes and datatypes themselves. Functions and predicates are always used as prefix operators.

The following definition describes the syntax and semantics of J. We use an interpretation function \( \text{I} : J \times \text{ENV} \rightarrow \text{ENV} \) for defining the semantics of statements, where the first parameter is a language construct of J and \( \text{ENV} \) is the set of all variable environments: each environment is a function which maps variables from \( \text{VARS} \) (the set of variables) to values from \( \text{A_D} \). Moreover, the function \( \text{I_E} : J \times \text{ENV} \rightarrow \text{A_D} \) captures the semantics of expressions.

Given a variable environment \( \omega \in \text{ENV} \), the syntax and semantics of J is defined as follows:

- **Expressions:**
  - Every constant \( c \in A_D \) is a valid expression: \( \text{I_E}(c, \omega) = c \)
  - Every variable \( x \in \text{VARS} \) is a valid expression: \( \text{I_E}(x, \omega) = \omega(x) \)
  - If \( f \) is either a function \( F_D \) or a predicate \( P_D \) and \( e_1, \ldots, e_k \) are expressions, then \( f(e_1, \ldots, e_k) \) is a valid expression. \( \text{I_E}(f(e_1, \ldots, e_k), \omega) = f(I_E(e_1, \ldots, e_k), \omega) \)

- **Statements:**
  - If \( x \in \text{VARS} \) is a variable and \( e \) is a valid expression, then \( x = e \) is a valid statement. \( I(x = e, \omega) = \omega' \) where \( \forall y \in \text{VARS} : \ y \neq x \rightarrow \omega'(y) = \omega(y) \) and \( \omega'(x) = I_E(e, \omega) \).
  - If \( e \) is a valid expression and \( s_1 \) and \( s_2 \) are block statements, then \( if \ e \ s_1 \ else s_2 \) is a valid statement. \( I(if \ e \ s_1 \ else s_2, \omega) = I(s_1, \omega) \) if \( I(e, \omega) = \text{true} \) and \( I(s_2, \omega) \) otherwise.

\[ x \geq 0 \land y \geq 0 \] // PRE-CONDITION
1. \( i = 0; \)
2. \( r = 0; \)
3. while (\( i < x \)) {
   \[ r == i \cdot y \] // INVARIANT
   4. \( r = r + \ldots \) assignment statement which calls the \( \Phi \) function.
   For example, the corresponding SSA form of the program
   statement and the changes to the semantics of exception;
   \( \Gamma : \) introduce a function \( \Gamma \) in his Alloy system for verification.
2. \[ \llbracket \{ 9 \} \rrbracket \] brings similar arguments and introduces the hypothesis
   test cases with a limited number of iterations and the number of steps is known in advance. As we debug a program using a given test case, we can simply execute the program for this test case in order to determine the maximum number of iterations. Note that in practice test cases with a limited number of iterations are often sufficient for detecting and locating faults. Jackson [9] brings similar arguments and introduces the small scope hypothesis in his Alloy system for verification.

We formalize the bounded compilation from \( \text{while} \)-statements into nested \( \text{if} \)-statements. For this purpose we introduce a function \( \Gamma : J \times \mathbb{N} \rightarrow J \):

\[
\Gamma(\text{while } e s_1, n) =
\begin{cases}
\text{if } e s_1 \Gamma(\text{while } e s_1, n-1) & \text{if } n > 0 \\
\text{if } e \{ \text{exception}; \} & \text{otherwise}
\end{cases}
\]

For all other statements \( \Gamma \) is assumed to be the identity function. We introduce the statement \( \text{exception}; \) because it allows us to detect if the assumed number of iterations is exceeded for a given test case. We define the semantics of this statement and the changes to the semantics of \( J \) as follows:

\[ I(\text{exception}, \omega) = \bot \] and for all statements \( x: I(x, \bot) = \bot \). Hence, if \( \text{exception} \) is reached, no new value can be derived except \( \bot \). Note that this definition slightly redefines the interpretation function \( I : J \times (ENV \cup \{ \bot \}) \mapsto ENV \cup \{ \bot \}. \)

Given \( \Gamma \) and the other definitions we state that the loop-free program is equivalent to the original program:

**Lemma 1** Let \( \Pi \in J \) be a program and \( n \in \mathbb{N} \). Then \( \Pi \) and \( \Pi_{LF} = \Gamma(\Pi, n) \) behave equal wrt a given input environment \( \omega \in ENV \) iff no exception is reached, i.e., \( I(\Pi_{LF}, \omega) \neq \bot \).

This lemma follows from the obvious fact that \( I(\Pi_{LF}, \omega) \neq \bot \Rightarrow I(\Pi_{LF}, \omega) = I(\Pi, \omega) \) holds.

The loop-free variant of the program from Fig. 1 is depicted in Fig. 2 for \( n = 2 \). It can be used for all cases where \( x \in \{0, 1, 2\} \) without leading to a different behavior compared to the original program.

\[ i = 0; \]
2. \( r = 0; \)
3. \( \text{if } (i < x) \{ \)
   4. \( r = r + y; \)
   5. \( i = i + 1; \)
   6. \( \text{if } (i < x) \{ \)
      7. \( r = r + y; \)
      8. \( i = i + 1; \)
      9. \( \text{if } (i < x) \{ \text{exception}; \} \)
   10. \}
11. \}

**Figure 2**: Loop unrolling of the program from Fig. 1 for \( n = 2 \) iterations. The annotations are omitted here.

### 2.2 Loop-free programs

Since loops cannot be directly represented as a constraint system, we transform the original program to a loop-free variant and prove its equivalence.

If the body of a \( \text{while} \)-loop is executed at most once, then the behavior corresponds to the single execution of a conditional statement. In general, if a \( \text{while} \)-condition is fulfilled, the statements in the block are executed and afterwards the condition is evaluated again. Hence, programs can be compiled into their loop-free equivalent if the number of steps is known in advance. As we debug a program using a given test case, we can simply execute the program for this test case in order to determine the maximum number of iterations. Note that in practice test cases with a limited number of iterations are often sufficient for detecting and locating faults. Jackson [9] brings similar arguments and introduces the small scope hypothesis in his Alloy system for verification.

We formalize the bounded compilation from \( \text{while} \)-statements into nested \( \text{if} \)-statements. For this purpose we introduce a function \( \Gamma : J \times \mathbb{N} \rightarrow J \):

\[
\Gamma(\text{while } e s_1, n) =
\begin{cases}
\text{if } e s_1 \Gamma(\text{while } e s_1, n-1) & \text{if } n > 0 \\
\text{if } e \{ \text{exception}; \} & \text{otherwise}
\end{cases}
\]

For all other statements \( \Gamma \) is assumed to be the identity function. We introduce the statement \( \text{exception}; \) because it allows us to detect if the assumed number of iterations is exceeded for a given test case. We define the semantics of this statement and the changes to the semantics of \( J \) as follows:

\[ I(\text{exception}, \omega) = \bot \] and for all statements \( x: I(x, \bot) = \bot \). Hence, if \( \text{exception} \) is reached, no new value can be derived except \( \bot \). Note that this definition slightly redefines the interpretation function \( I : J \times (ENV \cup \{ \bot \}) \mapsto ENV \cup \{ \bot \} \).

Unique variable names can be easily obtained by adding an index at the end of the name. However, although converting programs comprising only assignment statements is straightforward, it is more difficult to convert programs with loops or conditional statements. As we transform loops to nested \( \text{if} \)-statements, we only need to consider the conditional statements.

The idea behind the conversion of conditional statements is as follows. The value of the condition is stored in a new unique variable. The \( \text{if} \)- and the \( \text{else} \)-branches are converted separately. In both cases the conversion starts using the indices of the variables already computed. Both conversions deliver back new indices of variables. In order to get a value for a variable we have to select the last definition of a variable from the \( \text{if} \)- and \( \text{else} \)-branch depending on how the \( \text{if} \) condition evaluates. This selection is done using a function \( \Phi \). Hence, for every variable which is defined in the \( \text{if} \)- or the \( \text{else} \)-branch we have to introduce a selecting assignment statement which calls the \( \Phi \) function.

For example, the corresponding SSA form of the program fragment

\[
\begin{align*}
&\text{Figure 1: A program for computing the product of two natural numbers.} \\
&\text{- If } e \text{ is a valid expression and } s_1 \text{ is a block statement, then while } e \ s_1 \text{ is a valid statement,} \\
&\quad I(\text{while } e \ s_1, \omega) = I(\text{while } e \ s_1, I(s_1, \omega)) \text{ if } I(e, \omega) = \text{true} \text{ and } \omega, \text{ otherwise.} \\
&\text{- If } s_1, \ldots, s_k \text{ are valid statements, then} \\
&\quad \{ s_1[s_2[\ldots[s_k] \ldots] \} \text{ and } \{ \} \text{ are block statements.} \\
&\quad I(\{ s_1[s_2[\ldots[s_k] \ldots] \}, \omega) = I(\{ \ldots s_k \}, I(s_1, \omega)), \text{ I(}{}{\omega}) = \omega. \\
\end{align*}
\]
1. \( i_0 = 0; \)
2. \( r_0 = 0; \)
3. \( \text{var}_e = (i_0 < x_0); \)
4. \( r_1 = r_0 + y_0; \)
5. \( i_1 = i_0 + 1; \)
6. \( r_2 = \phi(r_1, r_0, \text{var}_e); \)
7. \( i_2 = \phi(i_1, i_0, \text{var}_e); \)

Figure 3: The SSA form of the loop-free variant of the program from Fig. 1 (for one iteration).

\[
\text{if } \epsilon \{ \ldots \ x = \ldots \} \text{ else } \{ \ldots \ x = \ldots \}
\]

is given as follows:

\[
\text{\text{var}_e = \epsilon};
\]

\[
\ldots
\]

\[
x_i = \ldots
\]

\[
\ldots
\]

\[
x_j = \ldots
\]

\[
\ldots
\]

\[
x_k = \Phi(x_i, x_j, \text{var}_e);
\]

where we define the function \( \Phi \) as follows:

\[
\Phi(x, y, b) = \begin{cases} x & \text{if } b \\ y & \text{otherwise} \end{cases}
\]

The SSA representation for the program from Fig. 1 is depicted in Fig. 3. For brevity, only one iteration is considered. Moreover, note that in this conversion we ignore the exception statement and we denote the function \( \Phi \) by \( \phi \).

Obviously the transformation of loop-free programs into their SSA form does not have an influence on the actual behavior (apart from the variable renaming), i.e., \( \Pi_{LF} = \Pi_{SSA} \) without any restrictions.

**Lemma 2** Given a program \( \Pi \) and \( n \in \mathbb{N} \). The loop-free variant \( \Pi_{LF} = \Gamma(\Pi, n) \) of \( \Pi \) behaves equivalent to its SSA form \( \Pi_{SSA} \). I.e., if \( I(\Pi_{LF}, \omega) = \omega' \) and \( I(\Pi_{SSA}, \Omega) = \Omega' \) with \( \Omega(x, j) = \omega(x) \) for all variables \( x \in VARS \), then for all variables \( y \in VARS \) there must be a \( y_j \) such that \( \omega'(y) = \Omega'(y_j) \).

### 2.4 Constraint representation

The final step of the conversion is the compilation to a constraint satisfaction problem (CSP). A CSP \( (V, D, CO) \) is characterized by a set of variables \( V \), each variable having a domain \( D \), and a set of constraints \( CO \), where each constraint defines a relation \( R \) between variables. The variables occurring in a relation \( R \in CO \) are called the scope \( S_R \) of the relation. A solution of a CSP is an assignment of values to all variables which do not contradict any given constraint. For more information regarding CSPs we refer to Dechter [4].

The constraint representation of a program is extracted from its SSA representation which comprises only assignment statements. Let \( \Pi_{SSA} \) be a program in SSA form. Then the corresponding CSP \( CSP(\Pi_{SSA}) \) is constructed as follows:

- Every statement \( x = \epsilon \) can be converted to a relation \( R \) where the scope \( \{x_1, \ldots, x_n\} \) is equivalent to the set of variables used in expression \( \epsilon \). The relation \( R(x, i_1, \ldots, x_n) \) is defined as follows: For all \( \omega \in ENV \) with \( \omega(x_i) = v_i \); if \( (x = \epsilon, \omega) = \omega' \), then \( R(\omega'(x), v_1, \ldots, v_n) = \text{true} \), otherwise false.

Because of the construction of the conversion function between \( \Pi_{SSA} \) and its corresponding CSP \( CSP(\Pi_{SSA}) \) we are able to prove the equivalence of both representation with respect to behavior.

**Lemma 3** Given a program \( \Pi_{SSA} \) in SSA form and its corresponding CSP representation \( CSP(\Pi_{SSA}) \). For all \( \omega \in ENV \); \( I(\Pi_{SSA}, \omega) = \omega' \) iff \( \omega \cup \omega' \) is a solution of \( CSP(\Pi_{SSA}) \).

Using this lemma we can finally conclude that the whole conversion process is correct:

**Theorem 4** Given a program \( \Pi \in J \) and \( n \in \mathbb{N} \), the loop-free representation, the SSA form and the CSP representation of \( \Pi \) are equivalent under the given assumptions, i.e., \( \Pi = \Pi_{LF} = \Pi_{SSA} = CSP(\Pi) \).

This theorem follows directly from Lemma 1 to 3.

### Example

We illustrate the last conversion step using the program from Fig. 1. From the SSA form which is depicted in Fig. 3 we extract the following CSP representation:

- **Variables**: \( V = \{x_0, y_0, i_0, r_0, \text{var}_e, r_1, i_1, r_2, i_2\} \)
- **Domains**: \( D = \{D(x) = \mathbb{N} \mid x \in V\} \)
- **Constraints**:

\[
CO = \left\{ \begin{array}{ll}
i_0 = 0, r_0 = 0, \text{var}_e = (i_0 < x_0), \\
r_1 = r_0 + y_0, i_1 = i_0 + 1, \\
r_2 = \phi(r_1, r_0, \text{var}_e), \\
i_2 = \phi(i_1, i_0, \text{var}_e)
\end{array} \right\}
\]

### 3 Integrating Annotations

We first show by means of our running example that the integration of annotations into debugging can improve the diagnosis results by reducing the number of obtained diagnoses.

Intuitively, a simple debugging algorithm, which searches for single faults only, can be outlined as follows:

1. For every statement \( S \) of the original program:
   - (a) If \( S \) is an assignment statement of the form \( v = a \):
     replace \( S \) by a new statement \( v = ? \), where "?" denotes an unknown value.
   - (b) Otherwise \( S \) is either a conditional statement
     _if_ \( (\epsilon) \) or a loop statement _while_ \( (\epsilon) \): replace \( S \) by _if_ \( (? \) or _while_ \( (? \), respectively.
   - (c) Transform the modified program, together with a given test case, to a CSP.
   - (d) If the resulting CSP has a solution, then \( S \) is a single-fault diagnosis (i.e., a bug candidate).

Now suppose we modify Line 3 in Fig. 1 to

\[
\text{while } (i \leq x).
\]

If we apply the test case \( \{x = 0, y = 2, r = 0\} \), which leads to one iteration...
of the loop, the algorithm above yields the following single-fault diagnoses: \( \{ S_1, S_2, S_3 \} \), where \( S_i \) corresponds to Line \( i \) in the program. E.g., assuming \( S_2 \) (i.e.: \( x = 0 \)) to be faulty leads to the following CSP for one iteration. Note that it also contains the test case, which is highlighted by bold letters:

\[
CO = \begin{cases} 
  x_0 = 0, y_0 = 2, r_2 = 0, \\
  r_0 = ?, \\
  i_0 = 0, var_x = (i_0 \leq x_0), \\
  r_1 = r_0 + y_0, i_1 = i_0 + 1, \\
  i_2 = phi(i_1, i_0, var_x),
\end{cases}
\]

Clearly, for \( r_0 = -2 \) none of these constraints is violated; hence, \( S_2 \) is a diagnosis.

The integration of pre- and post-conditions is trivial: every pre- and post-condition is represented by a single constraint in the CSP. However, loop invariants must hold before and after each execution of the loop’s body. Hence, if only one loop iteration is considered, then the resulting CSP contains two constraints for every invariant. If we have two iterations, then we obtain 3 constraints for every invariant: 2 constraints related to the first iteration, and a third one related to the end of the second iteration. There is no need for a constraint related to the beginning of the second iteration, because it would be equal to the second constraint for the first iteration. In general, for one loop invariant and \( n \) iterations, we need \( n + 1 \) constraints.

For the example above, which assumes \( S_2 \) to be faulty, we obtain the following new constraints capturing the annotations in Fig. 1:

\[
CO = \begin{cases} 
  x_0 = 0, y_0 = 2, r_2 = 0, \\
  \ldots, \quad \text{[as above]} \\
  x_0 \geq 0, y_0 \geq 0, \quad \text{[pre-cond.]} \\
  r_0 = i_0 * y_0, \quad \text{[invariant]} \\
  r_1 = i_1 * y_0, \quad \text{[invariant]} \\
  r_2 = x_0 * y_0 \quad \text{[post-cond.]} 
\end{cases}
\]

This CSP has no solution: because \( var_x \) is true we conclude that \( r_2 = r_1 \) must hold. From the invariant \( r_1 = i_1 * y_0 \) we can derive that \( r_2 = 2 \), hence \( r_2 = 0 \). However, this contradicts the test case \( r_2 = 0 \). Therefore, the integration of the annotations eliminates the candidate \( S_2 \), and only \( S_1 \) and \( S_3 \) remain as diagnoses. The crucial point is that the annotations provide additional informations which, when regarded as constraint, can rule out diagnoses which otherwise would be possible.

4 Debugging using the CSP

4.1 Background: hypertree decomposition

The structure of a CSP can be represented by a hypergraph, a concept which generalizes the notion of graph. A hypergraph is a pair \((HV, HE)\), where \( HV \) is a set of vertices and \( HE \) a set of hyperedges. Each hyperedge is a non-empty subset of \( HV \); i.e., it may connect more than two vertices. In our case, \( HV = V \) where \( V \) denotes the set of variables of the CSP representation, and there is exactly one hyperedge \( ER \in HE \) for each constraint \( R \in CO \), and \( ER \) comprises all variables in the scope of \( R \).

Hypergraphs can be used to classify CSPs wrt the complexity of computing a solution. It is well known that CSPs whose hypergraph is acyclic can be solved in polynomial time [18]. Such hypergraphs can be directly represented as hypertrees.

However, even CSPs whose hypergraph contains cycles can be decomposed into a data structure called hypertree decomposition, which is a hypertree where each vertex corresponds to one or more edges of the original hypergraph, i.e., to one or more constraints of the CSP [6; 7]. Note that there may be several possible hypertree decompositions for a given CSP. Computing the solutions for the decomposed CSP involves a join operator, which is responsible for joining the constraints associated with one vertex of the hypertree decomposition. As this operator basically solves sub-problems of the original CSP, it is computationally very complex. The width of a given hypertree decomposition denotes the maximum number of constraints to be joined, and the hypertree width of a hypergraph is the minimum width over all possible hypertree decompositions for this hypergraph.

The authors of [7] state that any given CSP \( I \) and a hypertree decomposition of \( I \) with a hypertree width \( k \) can be solved in \( O(|I|^k \cdot \log|I|) \), where \(|I|\) corresponds to the input size of \( I \). I.e., for a fixed \( k \), any CSP whose hypertree width is bounded by \( k \) is tractable. In practice, by performing a hypertree decomposition one can only gain a computational advantage when the hypertree width is sufficiently small. The complexity of computing a hypertree decomposition is polynomial.

Note that there are also other structural decomposition methods, but Gottlob et al. have shown that the hypertree decomposition generalizes other important decomposition methods [7].

4.2 The debugging process

The CSP representation of a given program can be directly used for debugging, i.e., for locating bugs within the source code. For this purpose we employ the TREE* algorithm introduced in [15]. The algorithm requires an acyclic CSP which can be obtained from \( CSP(\Pi_{SSA}) \) by performing, in our case, a hypergraph decomposition (other decomposition methods could also be used). The combination of TREE* and the decomposition method is described in [16]. TREE* is based on Yannakakis’ algorithm [18] which computes solutions for acyclic CSPs in polynomial time.

TREE* is able to compute multiple-fault diagnoses with a minimal cardinality. The overall diagnosis time mainly depends on the complexity of the corresponding CSP that itself can be characterized by the hypertree width, which should be as small as possible.

The overall diagnosis process which we propose in this paper comprises the following steps:

1. SSA Conversion: We first do an ‘unrolling’ of all loops of the program \( \Pi \) and obtain the new program \( \Pi_1 \). Afterwards we convert the program \( \Pi_1 \) into its SSA representation \( \Pi_2 \). Note that this step also requires the integration of assertions, pre- and post-conditions, and invariants.
2. **The CSP’s hypertree**: From $\Pi_2$ we generate the constraint representation and its corresponding hypertree decomposition.

3. **Diagnosis**: For a given test case, which is encoded as a set of constraints, we apply the TREE* algorithm. The test case specifies the expected variable values at certain positions in the program in order to compute the diagnoses, i.e., bug candidates.

## 5 Experimental Results

As already explained, the hypertree width is an important metric for the complexity of debugging based on a constraint representation. Complex debugging problem have a large hypertree width. In general, problems with a hypertree width of more than 5 can be considered as harder problems from a practical perspective.

Hence, we computed the hypertree width for a set of programs. We used a standard PC with a Pentium 4 Dual Core 2.0 GHz CPU with 2 GB of RAM, running under Windows XP. We implemented the software for converting programs into CSPs which run as Eclipse plug-in.

For computing the hypertree and the hypertree width we relied on an implementation provided by [19] which employs the Bucket Elimination algorithm [21]. Note that this algorithm is an approximation algorithm, i.e., it does not always generate the optimal hypertree decomposition with a minimal width. However, as reported in [21], the algorithm which performs the optimal decomposition is very demanding w.r.t. computation time and memory and hence not suitable for practical use, and the Bucket Elimination algorithm in most cases provides better approximations than other known approximation algorithms.

The obtained results are summarized in Fig. 4. The table comprises the lines of code (LOC), the lines of code of the corresponding SSA form (LOC_SSA), the number of while-statements (#W), the number of if-statements (#I), the number of considered iterations (#IS), and the hypertree width (HW) for each program.

The hypertree width varies from 2 more than 30. Although the obtained results are only for small programs they allow us to conclude that debugging programs is a very hard problem from the computational point of view.

The hypertree width obviously depends on the number of unrollings. We are particularly interested on the impact of increasing the number of loop iterations on the hypertree width. The interesting question is whether the hypertree width reaches an upper bound when increasing the number of loop iterations. If this is the case, then solving the corresponding CSP is, from a theoretical point of view, tractable. We performed several experiments with a larger number of iterations. Fig. 5 depicts the hypertree evolutions of three different programs. It can be seen that in all of these cases the hypertree width apparently reaches an upper bound.

When comparing those results with previous research, it is interesting to note that the author of [20] claims that the hypertree width is never greater than 6 for sequential programs. However, the author only considers the control flow, whereas for fault localization purposes it is also necessary to take the data flow into account.

The question whether there is always an upper bound for the hypertree width when increasing the number of iterations is still an open research question which requires further investigation.

We also tested our implementation of the TREE* diagnosis algorithm on a set of programs in which we injected single-faults. For example, for the program *ArithmeticOp* (see Fig. 4) we obtained 1 (minimal cardinality) diagnosis, for *Multiplication* including invariants there were 3 diagnoses, and for *SwitchingGate* 3 diagnoses were computed. In all cases, the faulty statement was included in the set of obtained diagnoses. As a comparison, after removing the annotations from *Multiplication* we got 7 diagnoses. While the diagnosis time for *ArithmeticOp* and *Multiplication* was less than 0.2 sec, our implementation required about 44.4 sec for *SwitchingGate*.

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>LOC_SSA</th>
<th>#W</th>
<th>#I</th>
<th>#IS</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinSearch</td>
<td>27</td>
<td>40</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>BinSearch</td>
<td>27</td>
<td>112</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Binomial</td>
<td>76</td>
<td>82</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Binomial</td>
<td>76</td>
<td>1155</td>
<td>5</td>
<td>1</td>
<td>30</td>
<td>$\geq$30</td>
</tr>
<tr>
<td>Hamming</td>
<td>27</td>
<td>62</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Hamming</td>
<td>27</td>
<td>989</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>$\geq$14</td>
</tr>
<tr>
<td>ArithmeticOp</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Huffman</td>
<td>64</td>
<td>78</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Huffman</td>
<td>64</td>
<td>342</td>
<td>4</td>
<td>1</td>
<td>20</td>
<td>$\geq$12</td>
</tr>
<tr>
<td>Multiplication</td>
<td>10</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Multiplication</td>
<td>10</td>
<td>350</td>
<td>1</td>
<td>0</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>whileTest</td>
<td>60</td>
<td>88</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>whileTest</td>
<td>60</td>
<td>376</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Permutation</td>
<td>24</td>
<td>41</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Permutation</td>
<td>24</td>
<td>119</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Permutation</td>
<td>24</td>
<td>1231</td>
<td>3</td>
<td>1</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>Adder</td>
<td>63</td>
<td>70</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>Sampowers</td>
<td>21</td>
<td>33</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Sampowers</td>
<td>21</td>
<td>173</td>
<td>2</td>
<td>1</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Sampowers</td>
<td>21</td>
<td>1376</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>SwitchingGate</td>
<td>15</td>
<td>23</td>
<td>0</td>
<td>4</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>IscasC432</td>
<td>162</td>
<td>162</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>9</td>
</tr>
<tr>
<td>ComplexHT</td>
<td>12</td>
<td>30</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ComplexHT</td>
<td>12</td>
<td>370</td>
<td>1</td>
<td>0</td>
<td>30</td>
<td>17</td>
</tr>
<tr>
<td>ComplexHT</td>
<td>12</td>
<td>1076</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>17</td>
</tr>
</tbody>
</table>

Figure 4: The hypertree width for different sequential programs. For most programs we depict the results for different numbers of iterations.

## 6 Further Related Research and Conclusion

Collavizza and Ruehner [2] discussed the conversion of programs into constraint systems and their use in software verification. They also make use of the SSA form but their constraint representation is different from ours. Moreover, we focus on debugging and program analysis rather than on verification.

Various authors, e.g., [5; 14; 11; 12], have described models to be used for fault localization using model-based diag-

---

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Figure 5: The hypertree width as a function of the number of iterations when unrolling while-statements.

nosis. Almost all of those works make assumptions regarding the program’s structure, and they use abstractions which lead to the computation of many diagnosis candidates. Often they do not handle all possible behaviors at once. For example, some models consider only one execution trace; this prevents the diagnosis engine to eliminate certain diagnosis candidates. In our representation we consider all possible behaviors up to a certain limit. This should lead to a reduction of the number of diagnosis candidates.

In [10] the authors discussed the use of Hoare logic for model-based debugging which requires a Hoare logic calculus for computing diagnosis. We also use Hoare logic to specify invariants, pre- and post-conditions, but we integrate them within the constraints system associated to the program.

The main restriction of the language J, which we defined for presenting our approach, is the fact that it does not support procedure calls. However, the integration of procedures (even recursive ones) is straightforward, as shown in [22].

In this paper we introduced a methodology for compiling programs into their equivalent CSP representation. We proved that all steps of the compilation process are correct with respect to the computed variable values. We also integrate annotations like pre- and post-conditions and invariants into the constraint representation, and we showed that those additional specifications can improve the diagnostic precision by eliminating candidates. We further explained how the resulting CSP can be used for debugging using the TREE* algorithm, and we present results from our implementation which shows that our approach is valid.

Because the hypertree width has a major impact on the performance of debugging, we further presented an empirical study on the hypertree width of a set of programs. The results show that debugging programs is a computationally very hard problem. However, the results also indicate that for a given program, when increasing the number of considered loop iterations, the hypertree width may reach an upper bound. Future research should seek a proof that there is always an upper bound, given a specific program. If this is the case, then debugging such a program would be, from a theoretical point of view, tractable.

References


[22] Franz Wotawa, Mihai Nica. On the Compilation of Programs into their equivalent Constraint Representation. Paper waiting to be reviewed.


A Bayesian Approach to Learning in Fault Isolation

Anna Pernestål
Dept. Electrical Engineering
Linköping University
Sweden
annap@isy.liu.se

Hannes Wettig, Tomi Silander
Helsinki Institute for Information Technology
Finland
{wettig,tsilande}@hiit.fi

Mattias Nyberg
Dept. Electrical Engineering
Linköping University
Sweden
matny@isy.liu.se

Petri Myllymäki
Helsinki Institute for Information Technology
Finland
petri@hiit.fi

Abstract

Fault isolation is the task of localizing faults in a process, given observations from it. To do this, a model describing the relation between faults and observations is needed. In this paper we focus on learning such models both from training data and from prior knowledge. There are several challenges in learning for fault isolation. The number of data, as well as the available computing resources, are often limited. Furthermore, there may be previously unobserved fault patterns. To meet these challenges we take on a Bayesian approach. We compare five different approaches to learning for fault isolation, and evaluate their performance on a real application; the diagnosis of an automotive engine.

1 Introduction

We consider fault isolation, i.e. the task of localizing faults that are present in a process given observations from this process. To do this, a model of the relations between observations and faults is needed.

In many traditional methods for fault isolation, the model of the relations is given by physical knowledge about the process, and represented as a structure describing which faults that may affect each observation, and possibly also how [Kołbcz et al., 2004; Hanscher et al., 1992; Nyberg, 2005; Pulido et al., 2005]. We call such knowledge expert knowledge. In many applications there is also data available from the process. In the current work we investigate and compare different methods for learning the model for diagnosis from training data, and the possibilities to integrate them with the expert knowledge.

We are motivated by the problem of fault isolation in an automotive engine, and we have used a Scania diesel engine as source for training and evaluation data. In engine fault isolation there may be several hundreds of faults and observations. There will be fault patterns, i.e. faults or combinations of faults, from which there is no training data. Furthermore, training data is typically experimental, meaning that it is obtained by implementing faults, running the process, and collecting observations.

To meet the challenge of previously unobserved fault patterns we consider a Bayesian approach to learning for fault isolation. Within the Bayesian framework it is also possible to also take other background information and expert knowledge into account, and not rely blindly on the data. We consider five different model classes when learning from training data. They are all previously presented in the literature in different forms. We tailor these methods to incorporate the available background information. The methods we consider are Direct Inference (DI), Logistic Regression (LogR), Linear Regression (LinR), Naive Bayes (NB) and general Bayesian Networks (BN).

The main contribution of the current work is the investigation of Bayesian learning methods for fault isolation by comparing models from the five classes mentioned above together with appropriate methods for learning their parameters. We do the comparison by application and evaluation of the methods on real-world data. In order to do the investigation of learning methods, we first discuss the characteristics of the fault isolation problem in terms of probability theory, and present performance measures that are meaningful for fault isolation. Consecutively we show how the five methods can be adopted to the isolation problem. We apply them to the task of fault isolation in an automotive diesel engine.

Bayesian methods for fault isolation have been previously studied in literature. In these previous works it is generally assumed that the model of the relations between faults and observations is given [Schwall and Gerdes, 2002; Lerner et al., 2000; Sheppard and Kaufman, 2005], or can be derived from a physical model without using training data [Narasimhan and Biswas, 2007; Roychoudhury et al., 2006], and focus is on inference. In the current work on the other hand, we consider five different model classes, and focus on learning the models of the relations, i.e. both structure and parameters.

Previous works on learning models, and in particular parameters in the models, for fault isolation from data typically rely on pattern recognition methods described for example in [Bishop, 1995; Devroye et al., 1996], or machine learning methods in [Heckerman et al., 1995b]. Applications are for example found in [Lee et al., 2007; Sheppard and Kaufman, 2005]. These methods are applicable if there is sufficient training data available. Unfortunately, this is rarely the case in fault isolation, where the number of training samples often is limited, at least for rare and safety critical faults. Furthermore, there are often missing fault patterns in the data. In the current paper we take a Bayesian approach to learning
for fault isolation, which provides a sound method also in the case of missing data, and opens the possibility to take prior knowledge into account.

In [Pernestål and Nyberg, 2007] the problem of learning with missing fault patterns is discussed. In [Pernestål and Nyberg, 2007] training data is combined with fundamental methods for fault isolation described in [de Kleer and Williams, 1992; Reiter, 1992]. This approach is referred to as DI in the current work, and compared to the other four methods for learning.

The paper is structured as follows. We introduce notation, and give a brief introduction to Bayesian networks in Section 2. We formulate the diagnosis problem in terms of probabilities in Section 3. Therein we also define relevant performance measures. In Section 4 we briefly describe the five methods used, and in particular how they are applied to the diagnosis problem. Then we perform evaluating experiments and compare the results obtained in Section 5. Finally, in Section 6 we conclude the paper by summarizing our results and discussing future work directions.

2 Preliminaries

Before going into the details of each of the learning methods we introduce notation that will be used, and give a brief introduction to Bayesian networks.

2.1 Notation

The fault isolation problem can be formulated as a prediction problem, where the task is to determine the fault(s) present in a process, given a set of observations from the process. Let the faults be represented by the binary variables \( Y = \{Y_1, \ldots, Y_K\} \), where \( Y_k = 1 \) means that fault \( k \) is present, and let the observations be represented by the variables \( X = \{X_1, \ldots, X_L\} \), where each \( X_i \) is discrete or continuous. Generally, we use upper case letters to denote variables, and lower case letters to denote their values. Boldface letters denote vectors.

We write \( p(X = x) \) (or simply \( p(x) \)) to denote either probabilities, probability distributions and probability density functions. The meaning will be clear from the context.

2.2 Fundamentals of Bayesian Networks

Bayesian networks are directed acyclic graphs in which nodes represent random variables and arcs represent directed probabilistic dependencies among the variables. We use the same notation for both nodes and variables. A Bayesian network encodes the joint probability distribution over a finite set of variables \( \{X_1, \ldots, X_L\} \), and decomposes it into a sequence of conditional probability distributions, one for each variable.

More specifically, let \( \text{pa}(X_i) \) denote the parents of \( X_i \), and let \( \text{pa}(x_i) \) be a value (configuration) of \( \text{pa}(X_i) \). Then there is a conditional probability distribution \( p(x_i | \text{pa}(x_i)) \) for each variable \( X_i \). Nodes without parents are called root nodes, and their conditional probabilities are simply their prior probability \( p(x_i) \). The joint probability distribution of \( \{X_1, \ldots, X_L\} \) can be obtained by taking the product of all these conditional probability distributions:

\[
p(x_1, \ldots, x_L) = \prod_{i=1}^{L} p(x_i | \text{pa}(x_i)). \tag{1}\n\]

In Bayesian networks, both the presence of arcs, and their directions, as well as the absence of arcs encodes knowledge about dependencies and independences. In addition to the structure of dependencies characterized by the edges in the Bayesian network, it also includes all the distributions \( p(x_i | \text{pa}(x_i)) \). When we discuss learning in Bayesian networks, we mean learning both the structure and the probability distributions.

3 Bayesian Fault Isolation

We are now ready to state the fault isolation problem in probabilistic terms, and present relevant performance measures.

3.1 Problem Formulation

Except the current observation \( X \) from the process, we are also given a set of training data \( D \). Training data consists of samples \( \{y^n, x^n\}, n = 1, \ldots, N_T \), of pairs of fault and observation variables. The training data is collected by implementing faults and then collecting observations, meaning that training data is experimental. To evaluate the fault isolation methods we use a set \( E \) consisting of \( N_E \) samples. The evaluation data is collected by running the process without integrating with i (i.e. without implementing any faults but rather observing faults as they appear), meaning that evaluation data is observational. Furthermore, we assume that the fault isolation algorithm is triggered by a fault detector telling us that there must be at least one fault present in the process.

The structure of dependencies between the faults and observations has three basic properties, illustrated in the example Bayesian network of Figure 1. The first property is that faults are assumed to be a priori independent, i.e. that

\[
p(y) = \prod_{k=1}^{K} p(y_k | y_1, \ldots, y_{k-1}) \approx \prod_{k=1}^{M} p(y_k), \tag{2}\n\]

meaning that faults do not cause other faults to occur. Although not necessary for the methods in the current work, this is a standard assumption in many fault isolation algorithms [Hamscher et al., 1992], and it simplifies the reasoning in the following sections.

Second, faults may causally affect one or several of the observation variables introducing dependencies between faults and variables. A dependency between fault variable \( Y_k \) and observation variable \( X_i \) means that the fault may be visible in the observation.

The third property is that an observation variable \( X_i \) may be dependent on other observation variables. Dependencies between observation variables can arise due to several reasons. For example they can be caused by unobserved factors, such as humidity, driver behavior, and operation point of the process. These unobserved factors could be modeled using hidden nodes, but since they are numerous and unknown they are here approximated with dependencies between observation variables. This is more carefully discussed in [Pernestål et al., 2006].

In the current work we take a Bayesian view point on fault isolation. The objective is to find the probability that each fault is present given the current observation, the training data, and the prior knowledge \( I \), i.e. to compute the probabilities \( p(y_k | x, D, I) \), \( k = 1, \ldots, K \). The probability for a
Figure 1: A Bayesian network describing a typical fault isolation problem.

A fault $y_k$ can be found by marginalizing over all other faults $y_{-k} = (y_1, \ldots, y_{k-1}, y_{k+1}, \ldots, y_K)$.

$$p(y_k | x, D, I) = \sum_{y_{-k}} p(y_{-k}, y_k | x, D, I).$$  \hspace{1cm} (3)

Note that $(y_{-k}, y_k) = y$, and (3) means that we seek the conditional distribution $p(y | x, D, I)$. To simplify the notation we will not write out the prior knowledge $I$ explicitly in the equations.

Computing the conditional distribution $p(y | x, D)$ is generally difficult. Instead, we approximate it using a model $M$, for example a Bayesian network or a regression model. For each model we need method for determining the parameters of the model. This means that we compute probabilities

$$p(y | x, D, M) = p_{M(D)}(y | x),$$  \hspace{1cm} (4)

where we have introduced the notation $p_{M(D)}(y | x)$ to denote the distribution obtained from training data $D$ by using model $M$ and the parameters determined using the appropriate method. To simplify notation we write $p_{M}(y | x)$ when there is no risk for confusion which data that is used.

### 3.2 Performance Measures

To evaluate the different models to be used in Bayesian fault isolation, we use two performance measures: the logistic score and the percentage of correct classification.

The logistic score is a commonly used performance measure [Bishop, 1995; Mitchell, 1997]. The logistic score based on a set $E$ of evaluation data is it given by

$$\mu(E, M) = \frac{1}{N_E} \sum_{n=1}^{N_E} \log p_{M(D)}(y^n | x^n).$$  \hspace{1cm} (5)

The score $\mu$ measures two important properties of the fault isolation system: the ability to assign large probability mass to faults that are present, as well as the ability to assign small probability mass to faults that are not present. In the fault isolation problem the conditional probabilities for faults are often combined with decision theoretic methods for troubleshooting [Heckerman et al., 1995a], where optimal decision making requires conditional probabilities close to the generating distribution.

The second performance measure we use, percentage of correct classification, is not a proper scoring function. However, it is closely related to the 0/1-loss used for example in pattern classification [Bishop, 1995]. We define

$$\nu(E, M) = \frac{|C|}{N_E},$$  \hspace{1cm} (6)

where $C = \{ n : y_k^n = 1, k = \arg \max_{k'} p_{M(D)}(y^n | x^n) \}$, and $y_k^n$ denotes element $k$ in $y^n$. In words, $C$ is the set of all indicies where the underlying fault is assigned the largest probability when model $M$ is used, and the $\nu$-score is thus the fraction of cases in evaluation data where the underlying fault is correctly classified. In case of multiple faults present it suffices to assign highest probability to any of them. The $\nu$-score reflects the performance of the fault isolation system combined with the simple troubleshooting strategy “check the most probable fault first”.

### 4 Modeling Methods

In this section we briefly present the modeling methods used, i.e. the different models used and methods for determining the parameters therein. We carefully state all assumptions made, and describe the adjustments of each method to apply it to the isolation problem. However, we begin by describing two assumptions that need to be made for all methods except DI.

#### 4.1 Modeling Assumptions

All the methods considered in this paper – with the exception of DI – build separate models for each fault and thus assume independence among these. Before any training data is recorded the approximation corresponds to (2). Furthermore, since faults were inflicted in training data, the data does not include any information about co-occurrence of the faults. However, when we build separate models for each fault, we also make a stronger assumption, namely that the faults remain independent given the observations.

$$p(y | x) = \prod_{k=1}^{K} p(y_k | x, y_1, \ldots, y_{k-1}) \approx \prod_{k=1}^{K} p(y_k | x)$$  \hspace{1cm} (7)

This approximation is (after applying Bayes’ rule and canceling terms) equivalent to

$$p(x | y_1, \ldots, y_K) \approx \frac{1}{p(x)^{K-1}} \prod_{k=1}^{K} p(x | y_k)$$  \hspace{1cm} (8)

In (8) $p(x)$ is a normalization constant, and the equation means that the observation $x$ is dependent on each fault $y_k$, but this dependency is assumed to be independent of all other faults $y_{k'}, k' \neq k$. In other words, we assume no “explaining away” effect [Jensen, 2001]. The explaining away effect can be understood as follows. Consider Bayesian network with two faults $Y_1$ and $Y_2$ and two observations, where $X_1$ that is dependent on both faults and $X_2$ is dependent on $Y_2$ only. Assume that observation $X_1$ indicates that there is a fault present (we say that $X_1$ “alarms”). Then both faults $Y_1$ and $Y_2$ are potential explanations. Now, assume that we learn that fault $Y_2$ is present (for example by observing that $X_2$ alarms), then fault $Y_2$ is likely to be the explanation of the alarm $X_1$ also. Since $X_1$ is explained by fault $Y_2$, fault $Y_1$ becomes less probable. The presence of $Y_2$ have explained away $Y_1$ through the observation $X_1$. 

---

Table 1: An example of an FSM

<table>
<thead>
<tr>
<th>X₁</th>
<th>Y₁</th>
<th>Y₂</th>
<th>Y₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₂</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The explaining away effect occurs when there are unsealed colliders, i.e., common children of two or more nodes which are them self not connected. Looking at Figure 1 we observe ignoring explaining away is indeed is a strong assumption, since there are several unsealed colliders of the faults. However, since each fault is allowed to be dependent on all observations, the explaining away effect will be partially encoded in the direct dependencies between faults and observations.

Assumption (7) is primarily made for technical reasons, in order to be able to build separate models for each fault. However, it is often the case (as in the application of Section 5) that there is training data only from single faults. Using training data straight-forward this would lead to that we learn a strong dependence between the faults: if one fault is present, other faults are not. By approximation (7) this is avoided, and we do not learn dependencies that irrelevant.

From Section 2 we known that it is assumed that there is at least one fault present. Let Y > 0 denote that \( \sum_k y_k > 0 \), i.e. that there is at least one fault present. Similarly, let Y = 0 denote \( \sum_k y_k = 0 \), i.e. that there is no fault present. The knowledge that there is at least one fault present recouples the single fault methods introduced in (7), since in general we have

\[
p(y|x, Y > 0) \neq \prod_k p(y_k|x, Y > 0), \quad (9)
\]

To avoid this recoupling, we study the probability for the faults given the knowledge that at least one fault is present i detail. We have

\[
p(y|x, Y > 0, D) = \frac{p(Y > 0|y, x, D)p(y|x, D)}{p(Y > 0|x, D)} = \begin{cases} 0 & y = 0, \\ \frac{p(y|x, D)}{1 - p(Y = 0|x, D)} & y = l \neq 0. \end{cases} \quad (10)
\]

In the current paper we ignore the fact that at least one fault is present during the learning phase and the single-fault models are trained individually. We then apply (10) in the evaluation phase.

4.2 Direct Inference

The first method for fault isolation that we present is the direct inference (DI). Similar to several previous fault isolation algorithms DI rely on prior knowledge about which observations may be affected by each fault [de Kleer and Williams, 1992; Reiter, 1992; Korbicz et al., 2004]. Such information is typically expressed in a so called Fault Signature Matrix (FSM). An example of an FSM is given in Table 1. In the FSM, a zero in position \((k, l)\) means that fault \( Y_k \) can never affect observation \( X_l \), while an \( X \) mean that \( Y_k \) may affect observation \( X_l \). DI aims at combining the information from the FSM with the training data available. Assuming that observations are binary and that the background information \( I \) contains the FSM. Then, under certain assumptions it can be shown [Mitchell, 1997; Pernestål and Nyberg, 2007] that

\[
p_{DI}(I|y, \alpha_{xy}) = \frac{\pi_0}{\alpha_{xy} + \sum_y p(y|x)\pi_y} \quad x \in \gamma, \quad (11)
\]

where \( \pi_0 \) is a normalization constant, \( \pi_{xy} \) is the count in training data \( D \) where the fault is \( y \) and the observation is \( x \), \( \alpha_{xy} \) is a parameter describing the prior belief in the observation \( x \) when the fault is \( y \). The parameters \( \alpha \) can be seen as hypothetical samples, which would have been obtained if our prior beliefs where true\(^1\), \( N_y = \sum_y \pi_{xy} \), and \( A_y = \sum_x \alpha_{xy} \). The sets \( \gamma \) are determined by the background information as described in [Pernestål and Nyberg, 2007].

The DI method is developed for sparse sets of training data, particularly when there is only training data from a subset of the fault patterns to isolate.

4.3 Bayesian Network Methods

When using Bayesian networks for prediction, we model the joint distribution \( p(y, x|\theta) \), where \( \theta \) are parameters describing the conditional probability distributions in the network. From the joint distribution, the conditional distribution for each of the faults \( y_k \) can be computed. As described in Section 4, we build one model for each fault, combine them using (7) and correct for the knowledge that there is no fault present by using (10), and then we can marginalize to obtain the probability for each fault. We consider two types of Bayesian networks: Naive Bayes and general Bayesian Networks.

**Naive Bayes**

In a **Naive bayes** network it is assumed that the observations are independent given the fault. This structure is exemplified in Figure 2. We assume this structure, and learn the parameters in the conditional probabilities using standard methods described for example in [Heckerman et al., 1995b]. Naive Bayes is is one of the most common methods used for Bayesian prediction and often performs surprisingly well [Devroey et al., 1996; Rish, 2001]. However, if there are strong dependencies between observations, the independence assumption made may introduce unnecessary large errors. For example, assume that two observations are identical. In this case, a better inference result may be obtained ignoring some of the observations that are strongly dependent. To alleviate this problem, we apply a variable selection according to an internal leave-one-out scoring function. This approach

---

\(^1\)the parameters \( \alpha \) are sometimes referred to as Dirichlet parameters, since a Dirichlet prior is used in the computations.
was first introduced in [Langley and Sage, 1994], where it is called selective naive bayes classifier. Let $V$ be the set of all subsets of the observations, let $V \in V$, and let $N_V$ be the Naive Bayesian network defined by $V$. We then choose the variable set $V^*$ according to

$$V^* = \arg \max_{V \in \mathcal{V}} \frac{1}{N_D} \sum_{n=1}^{N_D} \log p_{N_V}(D|(y^n, x^n)) \{y^n| x^n, \alpha\},$$

where $\alpha$ is the Dirichlet hyper-parameter for the NB model. The probabilities for fault $y_k$ is computed by

$$p_{N_V}(D)(y_k|x, \alpha),$$

where $\alpha$ is the Dirichlet hyper-parameter for the BN model.

**General Bayesian Network**

A natural extension of the naive Bayes model is to allow a more general structure for each fault, and learn both structure and conditional probabilities from the training data. However, it is known that the faults causally precede the observations. Therefore we restrict the possible structures to the ones where the fault node is a root node. This is the only constraint used. One Bayesian network (BN) was learned for each fault using a BDe score with an equivalent sample size parameter of 1.0 [Heckerman et al., 1995b]. For small systems ($< 30$ variables) learning can be performed using the exact algorithm in [Silander and Myllymäki, 2006], while for larger systems approximate methods, e.g. [Heckerman et al., 1995b; Mitchell, 1997; Russell and Norvig, 1995], can be used.

Let $B$ denote the Bayesian network learned using the BDe score. Then the probabilities for fault $y_k$ is computed by

$$p_B(D)(y_k|x, \alpha),$$

where $\alpha$ is the Dirichlet hyper-parameter for the BN model.

**4.4 Regression**

Fault isolation is a discriminative task, where we are to predict the fault vector $y$ given the observations $x$, i.e. to estimate the conditional probability of $y$

$$p(y|x, \theta) = \frac{p(y, x|\theta)}{\sum_y p(y, x|\theta)}. \quad (13)$$

It is well known [Ng and Jordan, 2002; Kontkanen et al., 2001; Friedman et al., 1997] that in such case it can be of great benefit to employ a discriminative learning method, that only learns the probabilities asked, instead of wasting training data to learn the joint data likelihood as in the Bayesian network methods of Section 4.3. Regression models form a family of such methods, and here we consider two classes of such: linear and logistic regression models.

**Linear Regression**

The most straight-forward regression method is linear regression, where each fault variable is assumed to be a linear combination of the observations plus a gaussian noise term,

$$y_k = w_k^T x + w_k 0 + e_k, \quad e \sim N(0, \sigma).$$

Here $w_k$, $w_k 0$, and $\sigma$ are parameters to be determined. This gives the probability distribution

$$p_{LinR}(y_k|x) = \frac{1}{Z} \exp \left( \frac{-\left( w_k^T x + w_k 0 - y_k \right)^2}{2\sigma^2} \right),$$

where $Z$ is a normalization constant. To determine the parameters we use the standard methods described for example in [Bishop, 1995]. For example,

$$w^* = \arg \min_{w} \sum_{n=1}^{N} (w_k^T x^n + w_k 0 - y_k^n)^2.$$

When the parameters $w^*$ are known, the parameters $\sigma$ and $Z$ can also be computed [Bishop, 1995].

**Logistic Regression**

Learning parameters to maximize (13) for a Bayesian network is known to be equivalent to logistic regression under the condition that no node can be a “bastard”, i.e. a common child of two variables that are not directly interconnected themselves. More formal definition and proofs can be found in [Roos et al., 2005]. In our case, this fact is guaranteed by assumption (7).

To start with, for each fault we learn a logistic regression model corresponding to a discriminative Naive Bayes classifier $^2$. Let $\alpha$ and $\beta$ be parameters in the logistic regression model, and define

$$p_{LogR}(Y_k = 1|x, \alpha, \beta) = \frac{\exp s(x, \alpha, \beta)}{\exp s(x, \alpha, \beta) + \exp -s(x, \alpha, \beta)}$$

where

$$s(x, \alpha, \beta) = \alpha + \sum_{l=1}^{L} x_l \beta_l.$$

When learning the parameters $\alpha$ and $\beta$, we use a smoothing term $c(\alpha, \beta)$ in the objective function. The smoothing function takes the place of a prior probability distribution for the parameters. To determine the smoothing term, we normalize training data such that

$$\sum_n x_i^n = 0 \text{ and } \max_n |x_i^n| = 1$$

Then, beginning with a uniform prior, $c'$, we pretend to have seen one vector of each fault at node $Y_k$ and two vectors of each fault with extreme values $\pm 1$ at each node $X_l$, with all other values unobserved. This amounts to a smoothing term

$$c'' = c' - 2\log(\exp(\alpha) + \exp(-\alpha)) - 4\sum_{l=1}^{L} \log(\exp(\beta_l) + \exp(-\beta_l)).$$

This smoothing term is problematic since it is flat near zero, leading to that no parameters will be exactly zero. In logistic regression many small parameters can make a large difference in the inference result, while they may be weakly supported. To avoid the flatness around zero $\log(\exp(z) + \exp(-z))$ was replaced by $|z|$ to obtain $c$ from $c''$. This is a good approximation away from zero, but forces unsupported parameters to zero, implicitly performing attribute selection.

For fault $y_k$ we search parameters that maximize

$$\log p_{LogR}(y_k|x, \alpha, \beta) + c(\alpha, \beta) =$$

$$= \sum_{n=1}^{N} \log p(y_k^n|x^n, \alpha, \beta) - 2|\alpha| - 4L |\beta||.$$
We do this by simple line search, one parameter at a time. Finally, we apply a variant of LogR, which we denote "LogR + weights", where training vectors are weighted according to their prior probabilities \( p(y_k) \). This is done since the training data and the evaluation data are known to have different distributions. The idea is to weight the training vectors in the objective function as to focus the optimization on areas of the data space more likely to be seen later on. The corresponding objective function for fault \( Y_k \) becomes

\[
\sum_{n=1}^{N_D} \log w_k p(y_k^n | x^n, \alpha, \beta) + c(\alpha, \beta)
\]

where the weight \( w_k \) is the prior \( p(y_k) \) divided by the observed relative frequency \( \# \{ n : y_k^n = y_k \} / N_D \).

5 Experiments

To evaluate the different modeling methods for fault isolation, we apply them to the diagnosis of the gas flow in a 6-cylinder diesel engine in a Scania truck. In automotive engines, sensor faults are one of the most common faults, and here we consider five faults that may appear in different sensors. The faults are listed together with their prior probabilities for single faults in Table 2.

### Table 2: The faults considered

<table>
<thead>
<tr>
<th>Fault</th>
<th>Description</th>
<th>( p(y_k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>exhaust gas pressure</td>
<td>0.4</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>intake pressure</td>
<td>0.13</td>
</tr>
<tr>
<td>( y_3 )</td>
<td>intake air pressure</td>
<td>0.057</td>
</tr>
<tr>
<td>( y_4 )</td>
<td>EGR vault position</td>
<td>0.13</td>
</tr>
<tr>
<td>( y_5 )</td>
<td>mass flow</td>
<td>0.057</td>
</tr>
</tbody>
</table>

As described in Section 4 the NB and DI methods perform best if not all observations are used. For both DI and NB we perform variable selection such that an internal logistic score is maximized. For DI, the best result is obtained by using only six of the observations. In NB between seven and 18 observations are used for each fault.

5.2 Results

In Table 3 the logistic score (\( \mu \)) and percentage of correct classification (\( \nu \)) are presented for the different methods. In addition to these we report the number of parameters used by each method. This is relevant, since for on-board fault isolation the computing and storage capacity is often limited. For comparison we also report the default which is obtained by simply using the prior probabilities given in Table 2.

### Table 3: Comparison of the methods

<table>
<thead>
<tr>
<th>method</th>
<th>( \mu )-score</th>
<th>( \nu )-score</th>
<th>#pars</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>-1.088</td>
<td>0.781</td>
<td>106</td>
</tr>
<tr>
<td>NB-bin.</td>
<td>-1.340</td>
<td>0.748</td>
<td>293</td>
</tr>
<tr>
<td>NB-disc.</td>
<td>-1.044</td>
<td>0.843</td>
<td>335</td>
</tr>
<tr>
<td>BN-bin.</td>
<td>-1.297</td>
<td>0.782</td>
<td>287</td>
</tr>
<tr>
<td>BN-disc.</td>
<td>-1.398</td>
<td>0.840</td>
<td>1136</td>
</tr>
<tr>
<td>LinR</td>
<td>-1.839</td>
<td>0.834</td>
<td>130</td>
</tr>
<tr>
<td>LogR</td>
<td>-1.071</td>
<td>0.829</td>
<td>46</td>
</tr>
<tr>
<td>LogR+weights</td>
<td>-0.953</td>
<td>0.829</td>
<td>44</td>
</tr>
<tr>
<td>default</td>
<td>-1.738</td>
<td>0.592</td>
<td>5</td>
</tr>
</tbody>
</table>

Considering the \( \mu \)-score, we see that among the four best methods in Table 3 three are discriminative and learn the conditional distribution instead of the joint distribution. Furthermore, LogR with training sample weighting performs best on this data in logistic score sense, while using a small number of parameters. Surprisingly the weighting trick has made quite a difference and LogR without weights is outperformed by NB-disc. NB performs better when it is fed with discretized observations instead of binary, while for BN the effect is reversed. Clearly the discretized data contain more information, but it seems that in more complex Bayesian networks the conditional probability tables grow too large, and there is not enough training data to learn them accurately. In DI good results are obtained by exploiting prior knowledge in terms of that some faults never cause an observation to pass certain thresholds.

Measured by the \( \nu \)-score the relative differences between the methods is smaller. This score favors the regression models and the Bayesian methods using discrete data.

Table 4 compares the logistic scores of the predictions given for the single faults by DI and LogR+weights. Note that because of inequality (9) the columns do not sum to the corresponding entries in Table 3. Both methods (as all others) have most trouble with isolating faults \( y_1 \), \( y_2 \) and \( y_4 \), the ones appearing simultaneously in evaluation data, but not in the same bin; and discretized using \( k \)-means clustering [Hartigan, 1975] with \( k = 4 \). DI is applied to the discrete data. NB and BN are run both on discrete and binary data. The regression methods LinR and LogR are applied to the continuous data.
Table 4: Comparison of DI and LogR on single faults

<table>
<thead>
<tr>
<th>Fault</th>
<th>$\mu_{DI}$</th>
<th>$\mu_{LogR+w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>-0.346</td>
<td>-0.385</td>
</tr>
<tr>
<td>$y_2$</td>
<td>-0.324</td>
<td>-0.287</td>
</tr>
<tr>
<td>$y_3$</td>
<td>-0.087</td>
<td>-0.008</td>
</tr>
<tr>
<td>$y_4$</td>
<td>-0.334</td>
<td>-0.294</td>
</tr>
<tr>
<td>$y_5$</td>
<td>-0.177</td>
<td>-0.133</td>
</tr>
</tbody>
</table>

training data. This gives evidence for explaining away being important in this problem. Figure 3, in which the probabilities for each fault using LogR + weights are plotted, shows this in more detail. In the Figure we have ordered the evaluation data such that the right-most samples have multiple faults, visualizing that the double faults are most difficult to predict.

6 Conclusions

We have considered the problem of fault isolation in an automotive diesel engine, and discussed the special characteristics of this problem. There is experimental training data available which is distributed differently from what we expect to see in the real-world setting. In particular, evaluation data consists partly of previously unseen fault patterns. In addition there is prior knowledge available about which faults may affect each observation, and also the knowledge that at least one fault is present.

We have studied different Bayesian and regression approaches to combine this by nature heterogeneous information into probability distributions for the faults conditioned on given observations. We have compared the performance of the methods using real-world data, and have found that on the application studied the discriminative logistic regression method to perform best. Among the methods that perform well we have also found the naive Bayes classifier and the direct inference method.

One of the clearest implications of this work is that all methods have difficulties with handling unobserved fault patterns. Unfortunately, unobserved patterns are common in fault isolation, so this problem should be tackled in future work. The four methods where one model is build for each fault, let the explaining away effect be present only through observations. However, this explaining away effect can possibly be helpful when diagnosing unseen patterns. Furthermore, it is crucial to include background information in the learning phase whenever it is available.

In our work to come we will apply the methods do several different applications in diagnosis to study if the results presented here are general. We will investigate models capable of both explaining away and taking prior knowledge into account, while providing an efficient inference procedure, as on-board computers offer very limited resources. We expect further improvement of performance is possible.

7 Acknowledgments

This project is partly financed by Scania CV AB and the Intelligent Vehicle Safety System (IVSS) program in Sweden.

![Figure 3: The predicted probability for the different faults given by LogR+w. Evaluation data is ordered after their fault patterns. The true fault is marked with a solid line.](image)

References


[Greiner and Zhou, 2002] Russel Greiner and Wei Zhou. Structural Extension to Logistic Regression: Discrimina-
A Bayesian Approach to Learning in Fault Isolation


Comprehensive Diagnosis of Continuous Systems Using Dynamic Bayes Nets
Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos
Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN, 37235, USA
Email: {indranil.roychoudhury, gautam.biswas, xenofon.koutsoukos}@vanderbilt.edu

Abstract
Fault diagnosis is essential for guaranteeing safe and reliable operation of complex engineering systems. Our work focuses on diagnosis of parametric faults in the components of dynamic systems, whose temporal profile can be categorized as incipient (slow) or abrupt (fast). The diagnosis of abrupt and incipient faults using qualitative approaches is challenging, since in many situations, these faults produce similar qualitative effects. Quantitative estimation methods may provide more discriminative power, but these approaches can be computationally infeasible for large systems with nonlinearities and complex dynamics. In this paper, we combine a qualitative fault isolation scheme with a Dynamic Bayes net-based particle filtering approach for the comprehensive diagnosis of incipient and abrupt faults in continuous systems. We also present experimental results to demonstrate the effectiveness of our approach when applied to a two-tank system.

1 Introduction
Detection, isolation, and identification of faults in system components is essential for guaranteeing safe, reliable, and efficient operation of complex engineering systems. Some of these faults are attributed to degradations, and modeled as incipient faults, i.e., slow drifts in system parameter values over time. Other faults manifest as quick changes in component parameter values and are modeled as abrupt faults, i.e., changes in parameter values that are fast in comparison to the system dynamics, and approximated as a step change.

In the past, we have successfully diagnosed abrupt faults using qualitative schemes [Mosterman and Biswas, 1999]. However, qualitative diagnosis schemes for both incipient and abrupt faults may suffer from the ambiguity problem, i.e., the inability to discriminate among fault hypotheses. Quantitative approaches produce more precise diagnoses, but, for large systems with complex dynamics, these quantitative approaches can be computationally expensive. In this paper, we extend our earlier work [Roychoudhury et al., 2006] to the comprehensive diagnosis of both incipient and abrupt faults in continuous dynamic systems through the integration of our qualitative diagnosis scheme [Mosterman and Biswas, 1999] with an Dynamic Bayes net-based particle filtering approach.

Dynamic Bayes Nets (DBNs) exploit the conditional independence among variables to provide a compact and factored representation of a dynamic system and allow arbitrary uncertainty models of the dynamic process and measurement [Murphy, 2002]. Hence, DBN-based tracking approaches need not conform to the restrictive assumption of normal distributions for noise and modeling errors. DBN schemes have been developed for several fault diagnosis problems [Lerner et al., 2000; Murphy, 2002]. This, however, makes the calculation of posterior probabilities computationally expensive, as, in many cases, no analytic closed form solutions for these probabilities exist. Particle filters (PFs) are a state-of-the-art inference mechanism using DBNs [Koller and Lerner, 2001] that help overcome the inability to derive analytic solutions for posterior probabilities. Moreover, for large systems, fairly significant gains in computational efficiency can be achieved by leveraging the sparseness and compactness of the DBN structures over traditional state-space based methods. Especially, in DBN models of faulty systems, the introduction of fault parameters typically results in very few additional links.

In our approach, we use PFs applied to the DBN model of the nominal system to generate estimates of nominal system behavior that are robust to measurement noise and modeling error. A statistically significant non-zero residual value implies a fault, and the qualitative fault isolation (Qual-FI) scheme generates and prunes fault hypotheses as measurement deviations are observed. Then, the quantitative fault isolation and identification (Quant-FII) scheme is invoked to further refine the fault hypotheses. A faulty DBN is generated for each remaining fault candidate by including the fault parameter as a stochastic variable in the DBN, and a separate PF scheme is run on each faulty DBN to track the faulty system behavior. As the Qual-FI scheme continues to refine its fault hypotheses, the PFs tracking the measurements using the inconsistent fault models are terminated. Also, if the measurements estimated by the PF applied to a particular fault model significantly deviates from the observed faulty measurements, that fault candidate is deemed inconsistent and removed from the set of possible faults. Eventually, the PFs using the true fault model converges to the observed faulty measurements, and estimates the value of the fault parameter. This efficient pruning of inconsistent fault hypotheses based
on the qualitative and quantitative analysis of measurements helps in fast diagnosis of the true fault.

The paper is organized as follows. Section 2 presents mathematical models of incipient and abrupt faults. Section 3 describes our diagnosis architecture. The different models used in our approach are described in Section 4. Section 5 discusses our fault detection approach. While Section 6 explains the fault isolation and identification scheme for incipient and abrupt faults in detail. Section 7 presents experimental results, and conclusions are presented in Section 8.

2 Incipient and Abrupt Faults

The mathematical models for incipient and abrupt faults used in our approach are defined below. In general, incipient and abrupt faults can be additive and multiplicative. However, in this paper, we focus on parameter multiplicative systems, which are hard to analyze because they directly affect the system dynamics.

2.1 Incipient Faults

An incipient fault is a slow change in a system parameter. Hence, we model this fault as a linear, additive, drift term, $\sigma_i$, added to the nominal component parameter value function, $p(t)$. Since incipient faults are slow changes, we approximate $d(t)$ as a linear function with a constant slope. Fig. 1(a) shows an incipient fault profile.

Definition 1 (Incipient fault) An incipient fault profile is characterized by a gradual, slow drift in the corresponding component parameter value. The temporal profile of a parameter with an incipient fault, $p^i(t)$, is given by:

$$p^i(t) = \begin{cases} 
    p(t) + d(t) & t \leq t_f \\
    p(t) + \sigma_i (t - t_f) & t > t_f,
\end{cases}$$

where $p(t)$ represents the nominal parameter value, $d(t)$ is a drift function, and $t_f$ is the time point of fault occurrence.

2.2 Abrupt Faults

An abrupt fault is modeled as an addition of a bias term, $b(t)$, to the nominal parameter value, $p(t)$. Typically, abrupt faults are very fast changes, and so, the bias term is modeled as an additive step function (see Fig. 1(b)). We assume the magnitude of this bias term to be constant.

Definition 2 (Abrupt fault) An abrupt fault profile is characterized by a fast change in the component parameter value.

The temporal profile of a parameter with an abrupt fault, $p^a(t)$, is given by:

$$p^a(t) = \begin{cases} 
    p(t) & t < t_f \\
    p(t) + b(t) = p(t) + \sigma_p^a & t \geq t_f,
\end{cases}$$

where $p(t)$ denotes the nominal parameter value, $b(t)$ is a bias term, and $t_f$ is the time point of fault occurrence.

3 Diagnosis Architecture

Our combined model-based approach for diagnosing abrupt and incipient faults, like traditional model-based diagnosis schemes [Gertler, 1998], has three primary components: (i) fault detection, (ii) fault isolation, and (iii) fault identification, as summarized below. The architecture of our diagnosis methodology is shown in Fig. 2.

Fault Detection: The dynamic nominal behavior of the system is tracked by a PF-based observer scheme [Koller and Lerner, 2001] based on a DBN model of the nominal system, i.e., the state variables and measurements made on the system are modeled as stochastic variables but the system parameters are considered to be deterministic and defined by their nominal values. Like other observer schemes, the PF generates estimates of the state variables, $\hat{x}(t)$, and measurements, $\hat{y}(t)$. The fault detector monitors each measurement residual, $r(t) = y(t) - \hat{y}(t)$, at each time step, where $y(t)$ is a measured variable at time $t$, and $\hat{y}(t)$ is the value of the measurement estimated by the PF. Ideally, $r(t) \neq 0$ should imply a fault and trigger the fault isolation scheme, but to accommodate measurement noise and modeling errors we employ a statistical testing scheme that balances detection sensitivity against false alarms, and a fault is detected if a non-zero residual is statistically significant.

Fault Isolation: Once a fault is detected, the fault isolation module is activated. Fault isolation is performed by running a Qual-FI scheme that uses the symbolic values of measurement deviations along with a Quant-FII approach that is based on a DBN-based PF scheme. The Qual-FI scheme for abrupt and incipient faults is described in Section 6.1. Once the number of fault hypotheses is less than a pre-defined threshold, $k$, or the Qual-FI scheme has run for $s$ steps, where $s$ is also a design parameter, we invoke the Quant-FII scheme.

The Quant-FI scheme starts with a separate DBN model for each fault candidate listed in the qualitative fault hypothesis set. The extended DBN includes the fault parameter as a stochastic variable in the DBN, and the corresponding PF tracks the faulty system parameters from the time of fault detection. If the system is diagnosable, the measurements estimated by the PF using the true fault model will converge.
to the observed measurements with minimum error. If, on the other hand, the measurements estimated by a PF using a particular fault deviates from the observed measurements, or the Qual-Fi scheme finds this fault candidate inconsistent, we terminate the tracking of the observations using this fault model and drop this fault candidate from our list of consistent hypotheses. We discuss this approach in greater detail in the subsequent sections.

**Fault Identification:** Fault identification involves the determination of the magnitude or extent of a fault. Our DBN-based Quant-FI scheme combines fault isolation and identification into the same PF-based tracking process.

## 4 Modeling

The bond graph (BG) [Karnopp et al., 2000] model of the system forms the core of our modeling framework. From BGs, we can systematically derive efficient models for diagnosis, the temporal causal graphs (TCGs) [Mosterman and Biswas, 1999] for qualitative fault isolation, and the DBNs, for detection and quantitative fault isolation and identification.

### 4.1 Bond Graphs

The bond graph modeling paradigm allows domain-independent, energy-based, topological modeling of physical processes. The nodes of a bond graph are energy storage (capacitors, $C$, and inertias, $I$); energy dissipation (resistors, $R$); energy transformation (gyrators, $GY$, and transformers, $TF$); and, energy source (sources of effort, $E$, and sources of flow, $Sf$) elements. Nonlinear systems are modeled by parameter values that are functions of other system variables. Bonds, drawn as half arrows, represent the energy exchange pathways between the bond graph elements. Two variables, effort, $e$, and flow, $f$, are associated with each bond, and their product, $e \cdot f$ defines the rate of energy transfer through the bond. Connections in the system are modeled by two idealized elements: 0- and 1-junctions. The junctions couple two or more elements based on the principles of conservation of energy and continuity of power. Therefore, at a 0-junction (or, a 1-junction), the efforts (or, flows) of all incident bonds are equal, and the sum of flows (or, efforts) is zero. Fig. 3(b) shows the BG of a simple two-tank system (shown in Fig. 3(a)). In the hydraulic domain, a flow represents the rate of flow of fluid, and effort represents the fluid pressure.

### 4.2 Temporal Causal Graph

The temporal causal graph (TCG) structure captures the causal and temporal relations between system variables [Mosterman and Biswas, 1999]. In our work, we derive the TCG systematically from the BG model. Fig. 3(c) shows the TCG for the two-tank system. The TCG represents a signal flow graph where the effort and the flow variables in the system model are nodes, and the direction and type of interaction between variables are captured as edges. Edges are derived from component constituent relations or junction constraints. For example, an edge describing the effort-to-flow relation of a resistance, $R$, is labeled $1/R$, since $f = e/R$. For a capacitor in integral causality, the flow-to-effort relation is labeled $dt/C$, where the $dt$ specifier implies a temporal edge, i.e., a change in the flow, $f$, affects the derivative of the effort, $e$. Junctions also impose direct ($+$), inverse ($-$), and equality ($=$) relations between variables.

### 4.3 Dynamic Bayesian Networks

A DBN is a two-slice Bayes net that not only captures the relations between system variables in any time slice $t$, but also captures the across-time relations between variables in time slice $t + 1$ and the previous time slice $t$ [Murphy, 2002]. The system variables, $(X, Z, U, Y)$, which represent the state variables, other hidden variables, input variables, and measured variables for the dynamic system, respectively, are all considered to be sampled from stochastic distributions. The dynamic state-space model is a discrete-time stochastic process that satisfies the first order Markov assumption. For the two-slice Bayes net, if an observed node is a function of a state variable, or an input variable, an intra-slice link is drawn from the state or input variable to that observed node. An inter-slice link, $x_t \rightarrow x'_{t+1}$, is drawn between two state variables, $x_t$ and $x'_{t+1}$, if the value of $x'$ at time $t + 1$ depends on the value of $x$ at time $t$. Nodes $x$ and $x'$ of the DBN may represent the same state variable, but at different time points. Similarly, an inter-slice link may also be drawn between an input variable at time $t$, and a state variable at time $t + 1$.

The DBN model for a system can be constructed from its TCG, as outlined in [Lerner et al., 2000]. First we identify the nodes, $N$, in the TCG that represent the state variables, system measurements, and inputs. Then for each of these nodes, $n \in N$, we create nodes $n_t$ and $n_{t+1}$ to denote the state of that variable at consecutive time points in the DBN. If the

![Figure 2: The diagnosis architecture.](image-url)
relation between any two TCG nodes, \( n, n' \in N \), is algebraic, links are constructed in the DBN from \( n_t \) to \( n'_t \), and \( n_{t+1} \) to \( n'_{t+1} \). On the other hand, if the relations between the two nodes has a delay, then a link is added in the DBN from \( n_t \) to \( n'_{t+1} \). The DBN models for the nominal and faulty systems are explained in detail below.

**Nominal System Model**

The DBN for the nominal system includes nodes corresponding to state variables, observed variables, and inputs. The system component parameters are assumed to be constant, or deterministic functions defined by the nominal system model. For example, as shown in Fig. 4(a), the DBN derived from the TCG of the two-tank system has the following stochastic variables at time \( t \): \( X_t = \{ e_2, e_7 \} \), the pressures at the bottom of tanks 1 and 2, respectively, and \( Y_t = \{ f_3, f_5, f_6 \} \), the outflows from tanks 1 and 2, and the flow between tanks 1 and 2, respectively. The input flow into tank 1, \( U_t = \{ f_1 \} \), and the component parameter values, \( C_1, C_2, R_1, R_{12}, \) and \( R_2 \) are deterministic variables. \( Z_t = \emptyset \), i.e., the two tank dynamic model requires no additional variables.

**Fault Models of the system**

For each fault candidate, a separate model is derived for tracking system behavior after fault occurrence. The fault model is generated by augmenting the nominal system model with an extra state variable, representing the fault parameter. For an abrupt fault hypothesis, the fault parameter corresponds to the bias term. For an incipient fault hypothesis, the fault parameter is the drift term as well as the fault parameter itself. This additional parameter accumulates the current value of the parameter corresponding to the incipient fault hypothesis.

The model of a two-tank system with an incipient \( R_{i1}^d \) fault includes the extra stochastic variable \( \sigma_{R_{i1}}^d \). We assume that the slope is constant, i.e., \( \sigma_{R_{i1}}^d (t+1) = \sigma_{R_{i1}}^d (t) \). The fault parameter \( R_{i1}^d (t) \) is included as an additional stochastic variable that evolves according to the equations \( R_{i1}^d (t+1) = R_{i1}^d (t) + \sigma_{R_{i1}}^d (t) \), and replaces all occurrences of \( R_{i1} \) in the nominal model. The DBN model for this fault is shown in Fig. 4(b).

The model of a two-tank system with an abrupt \( R_{i1}^{ad} \) fault includes the extra state variable \( \sigma_{R_{i1}}^{ad} \). We assume that the magnitude of this bias is constant, i.e., \( \sigma_{R_{i1}}^{ad} (t+1) = \sigma_{R_{i1}}^{ad} (t) \), where \( t \geq t_f \). We generate the faulty system model by replacing all occurrences of \( R_{i1} \) in the nominal model with \( (R_{i1} + \sigma_{R_{i1}}^{ad} (t)) \).

**5 Tracking and Fault Detection**

The basic idea of fault detection is to track nominal system behavior using PFs, and use a statistical hypothesis testing scheme to detect statistically significant non-zero residuals.

**5.1 Tracking**

Particle filtering is a popular scheme for estimating the true state of a system using DBNs [Koller and Lerner, 2001]. A PF is a sequential Monte Carlo sampling method for Bayesian filtering that approximates the belief state of a system using a weighted set of samples, or particles [Arulampalam et al., 2002; Koutsoukos et al., 2003]. Each sample, or particle, consists of a value for each state variable, and describes a possible state the system might be in. As more observations are obtained, each particle is moved stochastically to a new state, and the weight of each particle is readjusted to reflect the likelihood of that observation given the particle’s new state. The PF algorithm for DBNs is shown in Algorithm 1 [Koller and Lerner, 2001].

![Figure 3: Two-tank system models.](image_url)
We choose to run the PF scheme on the DBN models for tracking both nominal and faulty system behavior for several important reasons. Particle filtering applied to DBNs exploit the sparseness and compactness of DBNs (based on conditional independence of the variables) to provide computationally efficient solutions, especially because each observed random variable in a DBN typically depends on some, and not all state variables. The compactness of DBNs is especially noticeable in our DBN fault models, where each fault parameter typically affects the relation between a small number of state variables and measurements. Moreover, PFs are a good approximation of DBN propagation methods, when exact distributions cannot be computed analytically, especially for complex, nonlinear systems. Also, PFs can be implemented as anytime algorithms, and a trade-off between accuracy and time efficiency can be achieved by varying the number of particles [Dearden and Clancy, 2001]. Finally, the single-fault assumption allows for the decomposition of a complex multi-hypothesis isolation and identification problem into a set of simpler, single hypothesis PF-based tracking problems.

5.2 Fault Detection

The fault detector monitors each measurement residual and indicates the presence of a fault when a statistically significant non-zero fault is detected. In our work, the observer-estimated measurements are compared against the actual system measurements using a Z-test for difference in means for robust fault detection [Biswas et al., 2003]. The Z-test uses a sliding window scheme to compute the residual mean and signal variance. The choice of parameters for this scheme and the confidence level chosen for the Z-test determines the properties of the fault detection filter. These parameters also determine the tradeoff between false alarms and fast detection of faults.

6 Fault Isolation and Identification

Once a fault is detected, the Qual-FI scheme is triggered to generate the initial fault hypotheses and refine these hypotheses as additional measurement deviations are observed. The Qual-FI is run till either the fault hypotheses set is refined to a pre-defined size, $k$, a design parameter, which is typically set to 10% of the total number of fault hypotheses generated after a fault is detected, or a pre-specified simulation timesteps have elapsed, after which the Quant-FI scheme is invoked to isolate and identify the true fault. We need to choose $k$ and $s$ carefully because if $k$ is too large and $s$ is too small, the large number of remaining fault candidates would make the Quant-FI inefficient. On the other hand, if $k$ is very small, and $s$ is large, the isolation and identification task will be delayed. In the following, we describe the two isolation schemes in more detail.

6.1 Qualitative Fault Isolation

Our qualitative fault isolation scheme is based on deriving fault signatures from the TCG. A fault signature is a qualitative representation of the magnitude and higher order changes in a measurement caused by a fault [Mosterman and Biswas, 1999]. The qualitative deviations of the measurements are expressed using ‘+’, ‘−’, or ‘0’ symbols which denote that the observed measurement has increased from nominal, decreased from nominal, or is nominal, respectively. Since we are dealing with noisy measurement environments, we assume that only the magnitude and slope of a signal can be reliably measured at any point in time [Manders et al., 2000], and use these information to discriminate between faults. A typical fault signature of a fault $f$ for a measurement $m_i$ can be $(+-)$, which denotes a discontinuous increase followed by a gradual decrease in $m_i$ if fault $f$ occurs. A $(0−)$ signature of the same fault for another measurement, $m_j$, on the other hand, implies that the occurrence of $f$ will not generate any discontinuity in $m_j$, but cause $m_j$ to decrease gradually.

The detection of a fault triggers the symbol generation module for every measurement. A sliding window scheme, similar to the one used for fault detection, is applied to the measurement residuals, and the symbols for the magnitude and slope for each measurement is determined when they deviate [Manders et al., 2000].

Once a fault is detected, the hypothesis generation module of Qual-FI is invoked. This scheme propagates the changes in the parameters that are consistent with the observed deviation backwards along the TCG to generate the fault hypotheses. We combine the work reported in [Roychoudhury et al., 2006] with our previous work [Mosterman and Biswas,
Table 1: Selected fault signatures for the two-tank system.

<table>
<thead>
<tr>
<th>Fault</th>
<th>f3</th>
<th>f5</th>
<th>f8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1^{a\tau}$</td>
<td>+</td>
<td>0+</td>
<td>0+</td>
</tr>
<tr>
<td>$R_1^{i \tau}$</td>
<td>0</td>
<td>0−</td>
<td>0+</td>
</tr>
<tr>
<td>$R_{12}^{a\tau}$</td>
<td>0+</td>
<td>0+</td>
<td>0−</td>
</tr>
<tr>
<td>$R_{12}^{i \tau}$</td>
<td>0+</td>
<td>0+</td>
<td>−−</td>
</tr>
<tr>
<td>$R_3^{a\tau}$</td>
<td>0+</td>
<td>0+</td>
<td>0−</td>
</tr>
</tbody>
</table>

1999) to generate both abrupt and incipient fault candidates for every implicated parameter. For each abrupt fault candidate in the hypothesis set, a forward pass on the TCG yields the fault signatures, i.e., the effect of this fault on all remaining measurements [Mosterman and Biswas, 1999]. Since incipient faults cannot produce discontinuities in measurements, the fault signatures for incipient faults are of the form $(0\tau)$, where $\tau$ is the first non-zero symbol in the fault signature of an abrupt fault in the same system parameter and for the same direction of change. For example, as shown in Table 1, the signature of fault $R_1^{a\tau}$ for flow $f_3$ is $(0−)$, since that of fault $R_1^{a\tau}$ for flow $f_3$ is $(+++)$. As additional measurements deviate from nominal, the generated symbol deviations for these measurements are compared to the generated fault signatures, and if any fault signature is inconsistent with the observed symbol for that measurement, the fault candidate is dropped.

### 6.2 Quantitative Fault Isolation and Identification

Once the Qual-FI scheme discussed above refines the number of fault hypotheses to a pre-defined number, or $s$ timesteps have elapsed, the Quant-FI scheme is started. The Quant-FI performs both fault isolation and identification. For each fault candidate that remains at the time Quant-FI is initiated, we develop a faulty DBN system model, as explained in Section 4.3. We then run a particle filter for each of these DBN fault models, taking as input the measurements from the time of fault detection, $t_d$, as described in Algorithm 1. As more observations are obtained, only the PF using the correct fault model, ideally, should be converging to the observed measurements, while the observations estimated by the PFs using the incorrect fault models should gradually deviate from the observed faulty measurements. A fault candidate is dropped if: (i) the Qual-FI drops that fault candidate, or (ii) the measurements estimated by that fault model significantly deviates from the observed faulty measurements.

A Z-test is used to determine if the deviation of a measurement estimated by the PF from the corresponding actual observation is statistically significant. Since even the correct fault model will need some time before the particles start converging to the observed faulty values, we need to delay the invocation of the Z-tests for $s_d$ time steps, as otherwise, the Z-tests will indicate a deviation from observed measurements at the very onset for all fault models. We typically assume that the particles for the true fault model will converge to the observed measurements within $s_d$ time steps of its invocation.

Since the fault magnitude is included as a stochastic variable in every fault model, the magnitude of the true fault (i.e., the bias, $\sigma_p$, or, the slope, $\sigma_r$) is considered to be that estimated by the PF for the true fault model.

### 7 Experimental Results

In this section, we present some experimental results obtained by applying the proposed diagnosis approach to the two tank system shown in Fig. 3(a). In such hydraulic systems, the accumulation of sediment in the pipes are common examples of incipient faults. In addition, sudden blockages of pipes due to the entry of foreign objects in the pipes through the tanks can be examples of abrupt faults. These incipient faults are modeled as gradual increases in pipe resistances and represented as $R_1^{a\tau}$, $R_{12}^{a\tau}$, and $R_3^{a\tau}$. Abrupt faults are modeled as step increases in the pipe resistances, and represented as $R_1^{i \tau}$, $R_{12}^{i \tau}$, and $R_3^{i \tau}$. The flows, $f_3$, $f_5$, and $f_8$, through pipes $R_1$, $R_{12}$, and $R_2$, respectively, are the measured variables for our experiments. In our experiments, we assume all random variables, and the prior and conditional probabilities are Gaussian Normal. The mean and variance of each hidden variable is set based on empirical knowledge of the model. The means and variances of the observed variables, as well as the conditional probabilities, are functions of the estimated system parameters, and the parameters of distributions of the hidden variables. For the experiments below, we set $k = 5$ and $s = 300$ s.

System behavior is generated for a total of 400 time steps using a Matlab Simulink simulation model. According to standard practice, white Gaussian noise with zero mean and power $–40$ dBW is added to the measurements. The measurements are saved to a file, and then run through our fault diagnosis scheme (implemented in Matlab) to generate our experimental results.

#### 7.1 Experiment 1

We present a run of our diagnosis scheme for a specific fault scenario. An incipient fault in pipe $R_1$, $R_1^{a\tau}$, with $\sigma_{R_1} = 10\%$, is introduced at time step $t = 50$ s.

For this experiment, we consider measurements $f_5$ and $f_8$ only. The $R_1^{a\tau}$ causes both measurements to increase gradually from nominal. The fault detector signals an increase in $f_5$ at time step $t = 52$ s, followed by an increase in $f_8$ at time step $t = 55$ s. The symbol generator indicates that these changes are gradual, and not discontinuous. According to the fault signatures shown in Table 1, only $R_1^{a\tau}$ and $R_1^{i \tau}$ are consistent with the observed deviations.

Therefore, two separate PFs, one for $R_1^{a\tau}$ and $R_1^{i \tau}$ are initiated. The DBNs for the abrupt and incipient fault models are shown in Fig. 4(c) and Fig. 4(b), respectively. As more observations are obtained, the Z-tests indicate that the measurement estimates of the $R_1^{a\tau}$ PF significantly deviates from the observed faulty measurements. As soon as a Z-test indicates a deviation, the only remaining fault model consistent with the observed measurements, i.e., $R_1^{i \tau}$ is isolated as the true fault. While the true injected fault slope is $10\%$, the slope of the incipient fault, $\sigma_{R_1}^{i \tau}$, is estimated to be $12\%$ by the PF (see Fig. 5(c)). The measurements estimated by the PFs applied to the two fault models are shown in Fig. 5(a) and Fig. 5(b).
7.2 Experiment 2

We now present another experiment for a specific fault scenario. An abrupt fault, $R_3^{+,a}$, with $\sigma_R = 10\%$ is introduced in pipe $R_2$ at time step, $t = 50$ s. The measurements used in this experiment are flows $f_5$ and $f_8$.

The $R_3^{+,a}$ fault causes a gradual decrease in its flow $f_5$ from nominal. The fault detector signals this deviation at time step $t = 52$ s. The symbol generators output the symbols ‘+’ for the slope of residual of $f_5$, and ‘0’ for the residual magnitude. This is followed by a $0+$ deviation in flow $f_3$. As shown in Table 1, all faults but $R_1^{+,i}, R_2^{+,a}$, and $R_2^{+,i}$ are inconsistent with this observed decrease in magnitude of $f_3$ and $f_8$, and hence, the fault hypothesis set is $\{R_1^{+,i}, R_2^{+,a}, R_2^{+,i}\}$.

The Quant-FII scheme’s task is to both isolate and identify the magnitude of the true fault. We generate the DBN model for each of the three faults using the method described in Section 4.3, and run three PFs on these models, taking as inputs, only measurements at time points $t > 52$ s, the time of detection of the fault. Eventually, the Z-tests indicate that the observations estimated by the PFs applied to $R_1^{+,i}$ and $R_2^{+,i}$ have significantly deviated from the observed faulty measurements, correctly isolating fault $R_3^{+,a}$ as the true fault. The estimated measurements from fault models $R_1^{+,a}$, $R_2^{+,a}$, and $R_2^{+,i}$ are shown in Fig. 6(a), Fig. 6(b), and Fig. 6(c), respectively.

As we can see from Fig. 6(d), the PF identifies the fault magnitude to be about a $11\%$ step increase in $R_2$, while the true fault magnitude is $10\%$.

8 Discussion and Conclusions

PFs have been used extensively for system health monitoring and diagnosis of hybrid systems [Dearden and Clancy, 2001; Lerner et al., 2000]. The general approach involves the system to include discrete nominal and fault modes, with the evolution of the system in each discrete mode being defined using...
differential equations. The process of diagnosis then involves tracking the observed measurements using a PF that runs on the comprehensive system model till the particles eventually converge to a discrete fault mode. PFs have also been used to diagnose parametric incipient and abrupt faults [Koller and Lerner, 2001]. The usual approach for using PFs for diagnosis, however, cannot alleviate the problem of sample impoverishment, wherein particles in faulty state (with typically very low probability, and hence low weights) are dropped during the re-sampling process. Even though several solutions to this problem have been proposed [Verma et al., 2004], the diagnosis scheme still has to rank the different fault hypothesis based on their likelihoods, and report the most likely fault mode that justifies the observations the best. Our single fault assumption allows us to avoid the sample impoverishment problem by having a separate fault model for each fault hypothesis. Also, we do not rank the different fault hypotheses, and drop candidates based on their inability to track the observed faulty measurements.

In [Narasimhan et al., 2004], the authors propose an approach for combining look-ahead Rao-Blackwellised PFs (RBPFs) with the consistency-based Livingstone 3 (L3) approach for diagnosing faults in hybrid systems. In this approach, the nominal RBPF-based observer tracks the system evolution till a fault is detected, after which L3 generates a set of fault candidates that are then tracked by the fault observer (another RBPF). All the fault hypotheses are included in the same model, and tracked by the fault observer. In contrast, our approach executes the qualitative and quantitative fault isolation schemes in parallel, and uses separate fault models for each fault candidate.

In the future, we seek to investigate and solve a number of open issues and problems. First, we need to study the observability of the faulty models and their impact on diagnosis. For example, in the two tank system shown in Fig. 3(a), it is not possible to uniquely discriminate between \( R_{1}^{2+} \) and \( R_{2}^{1+} \) faults using measurements \( f_{3}, f_{5}, \) and \( f_{8} \). The problem of identifying the correct set of measurements such that the system is diagnosable as well observable, therefore, is an interesting research issue. Next, we wish to apply our diagnosis approach to a large real-world system, to analyze the scalability and efficiency of our methodology. Third, we need to develop systematic procedures for obtaining the values of design parameters \( k, s, \) and \( s_{l} \). Finally, we would like to improve the efficiency of our diagnosis approach by deriving reduced DBN models and running the PFs on these reduced-order models instead of on the entire system DBN model.

Acknowledgements

This work was supported in part by the National Science Foundation under Grant CNS-0615214 and NASA NRA NNX07AD12A.

References


Model-based Test Generation using Quantified CSPs

Martin Sachenbacher and Stefan Schwoon
Technische Universität München, Institut für Informatik
Boltzmannstraße 3, 85748 Garching, Germany
{sachenba,schwoon}@in.tum.de

Abstract

Testing is the process of stimulating a system with inputs in order to reveal hidden parts of the system state. In this paper, we consider finding input patterns to discriminate between different, possibly non-deterministic models of a technical system, a problem that was put forward in the model-based diagnosis literature. We analyze this problem for different types of models and tests with different discriminating strength. We show how the variants can be uniformly formalized and solved using quantified CSPs, a game-theoretic extension of CSPs. The results of the paper are (1) a map of the complexity of different variants of the testing problem, (2) a way to compute discriminating tests using standard algorithms instead of ad-hoc methods, and (3) a starting point to extend testing to a richer class of applications, where tests consist of stimulation strategies instead of simple input patterns.

1 Introduction

As the complexity of technical devices is growing, methods and tools to automatically check such systems for the absence or presence of faults become increasingly important. Diagnosability asks whether a certain fault can ever go undetected in a system due to limited observability. It has been shown how this question can be framed and solved as a satisfiability problem [Cimatti et al., 2003; Rintanen and Grastien, 2007]. Testing instead asks whether there exist inputs (test patterns) to stimulate a system, such that a given fault will always lead to observable differences at the outputs. For the domain of digital circuits with deterministic outputs, it has also been shown how this question can be framed and solved as a satisfiability problem [Larrabee, 1992; Brand, 2001].

In this paper, we consider constraint-based testing for a broader class of systems, where the models need not be deterministic. There are several sources for non-determinism in model-based testing of technical systems: in order to reduce the size of a model – for example, to fit it into an embedded controller [Williams et al., 2003; Sachenbacher and Struss, 2005] – it is common to aggregate the domain of continuous system variables into discrete, qualitative values such as ‘low’, ‘medium’, ‘high’, etc. A side-effect of this abstraction is that the resulting models can no longer be assumed to be deterministic functions, even if the underlying system behavior was deterministic. Another source is the test situation itself: even in a rigid environment such as an automotive test-bed, there are inevitably some variables or parameters that cannot be completely controlled while testing the device.

Notions of testing for non-deterministic models have been introduced in various areas. In the field of model-based reasoning with logical (constraint-based) system descriptions, Struss [Struss, 1994] introduced the problem of finding so-called definitely discriminating tests (DDTs), which asks whether there exist inputs that can unambiguously reveal or exclude the presence of a certain fault in a system, even if there might be several possible outputs for a given input. [Struss, 1994] provided a characterization of this problem in terms of relational (constraint-based) models, together with an ad-hoc algorithm to compute DDTs. Generating DDTs is a problem of considerable practical importance; the framework was applied to real-world scenarios from the domain of railway control and automotive systems [Esser and Struss, 2007]. Later work [Esser and Struss, 2007] extended the idea to systems modeled as automata, which, using a fixed bound of time steps, are unfolded into constraint networks such that the former algorithm can be applied. In the field of automata theory, [Alur et al., 1995; Boroday et al., 2007] have studied the analogous problem of generating distinguishing sequences, which asks whether there exists an input sequence for a non-deterministic finite state machine, such that based on the generated outputs, one can unambiguously determine the internal state of the machine.

In this paper, we give an overview and establish connections between these different notions of the testing problem, with a generalized form of constraint models serving as the glue. We show how the different variants can be conveniently formulated using quantified CSPs (QCSPs), an extension of CSPs to multi-agent (adversarial) scenarios. This leads to three contributions: first, we provide an overview of the complexity landscape of model-based testing for different combinations of discriminating strength and model types. For example, we observe that the problems of finding possi-
bly discriminating tests and finding definitely discriminating tests for logical models [Struss, 1994] have the same worst-case complexity, which is however less than those of finding distinguishing sequences for automata models. Second, we map the various test generation problems to QCSP formulas, which, instead of devising ad-hoc algorithms as in [Struss, 1994; Esser and Struss, 2007], enables to use off-the-shelf solvers in order to effectively compute tests. Third, we show that our QCSP (adversarial planning) formulation of testing can be straightforwardly extended to problems that require complex test strategies instead of simple input patterns, and thus go beyond the framework in [Struss, 1994; Esser and Struss, 2007].

2 Quantified CSPs (QCSPs)

In a constraint satisfaction problem (CSP), all variables are (implicitly) existentially quantified; we find an assignment for each of the variables that satisfies all constraints simultaneously. Quantified CSPs (QCSPs) are a generalization of CSPs that allow a subset of the variables to be universally quantified:

**Definition 1 (Quantified CSP)** A QCSP \( \phi = \langle Q, X, D, C \rangle \) has the form

\[
Q_1 x_1 \ldots Q_m x_m \cdot C(x_1, \ldots, x_n)
\]

where \( m \leq n \) and \( C \) is a set of constraints over the variables \( X = \{x_1, \ldots, x_n\} \) with domains \( D = \{d_1, \ldots, d_n\} \), and \( Q \) is a sequence of quantifiers where each \( Q_i \), \( 1 \leq i \leq m \), is either an existential (\( \exists \)) or a universal (\( \forall \)) quantifier.

**Definition 2 (Satisfiability of QCSP)** The satisfiability of a QCSP \( \phi = \langle Q, X, D, C \rangle \) is recursively defined as follows. If \( Q \) is empty then \( \phi \) is satisfiable iff the CSP \( \langle X, D, C \rangle \) is satisfiable. If \( \phi \) is of the form \( \exists x_1 Q_2 x_2 \ldots Q_n x_n \cdot C \) then \( \phi \) is satisfiable iff there exists a value \( a \in d_1 \) such that \( Q_2 x_2 \ldots Q_n x_n \cdot C \cap (x_1 = a) \) is satisfiable. If \( \phi \) is of the form \( \forall x_1 Q_2 x_2 \ldots Q_n x_n \cdot C \) then \( \phi \) is satisfiable iff for every value \( a \in d_1 \), \( Q_2 x_2 \ldots Q_n x_n \cdot C \cap (x_1 = a) \) is satisfiable.

Compared to the classical CSP framework, QCSPs have more expressive power to model particular aspects of real-world problems, such as uncertainty or other forms of uncontrollability in the environment. For example, in game playing, they can be used to find a winning strategy for all possible moves of the opponent.

There exist a number of solvers for quantified formulas, most of which use variants of search and local propagation, the dominating algorithmic approach for SAT/CSP problems. While such solvers are easy to implement because they build on existing technology, their performance often turns out to be not competitive with the alternative approach of expanding the problem into a classical instance (SAT/CSP) and using a SAT/CSP solver. However, it has recently been shown [Benedetti and Mangassarian, 2008] that more advanced algorithmic pre-processing and inference techniques, which usually do not pay off for classical problems, often work well for quantified problems, and can make QBF/QCSP approaches several orders of magnitude faster than classical approaches. It is therefore expected that QBF/QCSP solvers will see significant performance improvements in the future, similar to those that SAT/CSP solvers have undergone in the past.

3 Discriminating Tests for Logical Models

We briefly review the theory of constraint-based testing of physical systems as introduced in [Struss, 1994]. Testing attempts to discriminate between hypotheses about a system – for example, about different kinds of faults – by stimulating the system in such a way that the hypotheses become observationally distinguishable. Formally, let \( M = \bigcup_i M_i \) be a set of different models (hypotheses) for a system, where each \( M_i \) is a set of constraints over variables \( V \). Let \( I = \{i_1, \ldots, i_n\} \subseteq V \) be the subset of input (controllable) variables, \( O = \{o_1, \ldots, o_m\} \subseteq V \) the subset of observable variables, and \( U = \{u_1, \ldots, u_k\} = V - (I \cup O) \) the remaining, internal (uncontrollable and unobservable) variables. The goal of testing is then to find assignments to \( I \) (input patterns) that will cause different assignments to \( O \) (output patterns) for the different models \( M_i \).

**Definition 3 (Discriminating Tests)** An assignment \( t_j \) to \( I \) is a possibly discriminating test (PDT), if for all \( M_i \) there exists an assignment \( t_O \) to \( O \) such that \( t_I \land M_i \land t_O \) is consistent and for all \( M_j, j \neq i \), \( t_I \land M_j \land t_O \) is inconsistent. The assignment \( t_j \) is a definitely discriminating test (DDT), if for all \( M_i \) and all assignments \( t_O \) to \( O \), if \( t_I \land M_i \land t_O \) is consistent then for all \( M_j, j \neq i \), it follows that \( t_I \land M_j \land t_O \) is inconsistent.

For example, consider the (simplified) system in Fig. 1. It consists of five variables \( x, y, z, u, v \), where \( x, y, z \) are input variables and \( v \) is an output variable, and two components that compare signals \( (x,y) \) and add signals \( (u,z) \). The signals have been abstracted into qualitative values 'low' (L) and 'high' (H); thus, for instance, values L and H can add up to the value L or H, and so on. Assume we have two hypotheses about the system that we want to distinguish from each other: the first hypothesis is that the system is functioning normally, which is modeled by the constraint set \( M_1 = \{ f_{\text{diff}}, f_{\text{add}} \} \). The second hypothesis is that the adder is stuck-at-L, which is modeled by \( M_2 = \{ f_{\text{diff}}, f_{\text{add-stuck}} \} \). Then for example, the assignment \( x = L, y = H, z = L \) is a PDT for \( M \) (it leads to...

---

the observation \( v = L \) or \( v = H \) for \( M_1 \), and \( v = L \) for \( M_2 \), while the assignment \( x = L, y = H, z = H \) is a DDT for \( M \) (it leads to the observation \( v = H \) for \( M_1 \), and \( v = L \) for \( M_2 \)).

In the following, we restrict ourselves to the case where there are only two possible hypotheses, for example corresponding to normal and faulty behavior of the system. Note that DDTs are then symmetric: if a DDT discriminates \( M_1 \) from \( M_2 \), then it is also discriminates \( M_2 \) from \( M_1 \).

3.1 Characterizing PDTs and DDTs

We sketch how for logical (state-less models), finding PDTs and DDTs can be characterized as a game played between two opponents. The first player (3-player) tries to reveal the fault by choosing input values for which the two hypotheses yield disjunct observations. The second player (\( \forall \)-player) instead tries to hide the fault by choosing values for outputs or internal variables such that the two hypotheses yield overlapping observations. In the case of PDTs, he can choose values only for internal variables, whereas in the case of DDTs, he can choose values both for internal and observable variables. Both the 3-player and the \( \forall \)-player must adhere to the rules that they can only choose among values that are consistent with the model of the system, as not all values are possible in all situations (there might also be additional rules for the 3-player such that he can only choose among allowed inputs, but without loss of generality, we do not consider such restrictions here). The goal of the game is that exactly one hypothesis becomes true. Clearly, a PDT or DDT then exists if and only if the first player has a winning strategy.

Thus, the first form of testing in Def. 3, finding PDTs, corresponds to solving a QCSP and is captured by the formula

\[
\exists i_1 \ldots i_n \exists o_1 \ldots o_m \forall u_1 \ldots u_j \forall u_{j+1} \ldots u_k \cdot M_1 \land \neg M_2
\]

where \( u_1 \ldots u_j \) are the internal variables of \( M_1 \), and \( u_{j+1} \ldots u_k \) those of \( M_2 \). In analogy to (1), we can capture the second (stronger) form of testing, finding DDTs, by the following QCSP formula:

\[
\exists i_1 \ldots i_n \forall o_1 \ldots o_m \forall u_1 \ldots u_k \cdot M_1 \rightarrow \neg M_2
\]

4 Discriminating Tests for Automata Models

In this section, we extend the notion of hypotheses (models) to be discriminated from the case of logical (state-less) models to the more general case of dynamic models whose state can change over time, as for instance used in NASA’s Livingstone [Williams and Nayak, 1996] or MIT’s Titan model-based system [Williams et al., 2003]. This means that we are no longer searching for a single assignment to input variables, but rather for a sequence of inputs over different time steps. The following two definitions are adapted from [Cimatti et al., 2003]:

Definition 4 (Plant Model) A (partially observable) plant is a tuple \( P = (x_0, X, I, \delta, O, \lambda) \), where \( X, I, O \) are finite sets, called the state space, input space, and output space, respectively, \( x_0 \in X \) is the start state, \( \delta \subseteq X \times I \times X \) is the transition relation, and \( \lambda \subseteq X \times O \) is the observation relation.

For technical convenience, we henceforth assume that in all our plants \( \delta \) and \( \lambda \) are complete, that is, for every \( x \in X \) and \( i \in I \) there exists at least one \( x' \) such that \( (x, i, x') \in \delta \) and at least one \( o \in O \) such that \( (x, o) \in \lambda \).

The intuitive meaning of a plant is as follows: \( X \) is the set of states that the plant can assume, and the state is not revealed to the observer. When the plant is in state \( x \), input \( i \) will cause the state to change from \( x \) to \( x' \) provided that \( (x, i, x') \in \delta \). Moreover, it can emit the observable output \( o \) provided that \( (x, o) \in \lambda \).

We write \( \delta (x, i, x') \) for \( (x, i, x') \in \delta \), and \( \lambda (s, o) \) for \( (x, o) \in \lambda \). Note that a plant need not be deterministic, that is, the state after a transition may not be uniquely determined by the state before the transition and the input. Likewise, a plant state may be associated with several possible observations.

Definition 5 (Feasible Trace) Let \( P = (x_0, X, I, \delta, O, \lambda) \) be a plant, and \( \sigma = i_1, i_2, \ldots, i_k \in I^* \) be a sequence of \( k \) inputs and \( \rho = o_0, o_1, \ldots, o_k \in O^* \) a sequence of \( k + 1 \) outputs. Then \( (\sigma, \rho) \) is a feasible trace of \( P \) if there exists a sequence \( \sigma = x_0, x_1, \ldots, x_k \) of states such that \( \delta (x_{j-1}, i_j, x_j) \) for all \( 1 \leq j \leq k \) and \( \lambda (x_j, o_j) \) for all \( 0 \leq j \leq k \).

A plant represents a hypothesis about the actual behavior of the system under test. Given two such hypotheses \( P_1, P_2 \) we are interested in determining which of the hypotheses is true. To this end, our aim is to stimulate the system under test using a sequence of inputs, and observe the output sequence; if we find that the observed output can be generated by one plant but not by the other, we know which hypothesis is correct. In this sense we can extend the notion of discriminating tests (Def. 3) from static systems to dynamic systems (plants):

Definition 6 (Discriminating Test Sequences) Given two plants \( P_1 = (x_0, X, I, \delta, O, \lambda) \) and \( P_2 = (y_0, Y, I, \eta, O, \mu) \), a sequence of inputs \( \sigma \in I^* \) is a possibly discriminating test sequence (PDTS), if there exists a sequence of outputs \( \rho \in O^* \) such that \( (\sigma, \rho) \) is a feasible trace of \( P_1 \) but not of \( P_2 \). The sequence \( \sigma \) is a definitely discriminating test sequence (DDTS) for \( P_1 \) and \( P_2 \), iff for all sequences of outputs \( \rho \), it holds that if \( (\sigma, \rho) \) is a feasible trace of \( P_1 \), then it is not a feasible trace of \( P_2 \).
Notice that, due to our assumptions about completeness, for every input sequence $\sigma$ there exist sequences $\rho, \tau$ such that $(\sigma, \rho)$ is a feasible trace of $P_1$ and $(\sigma, \tau)$ is a feasible trace of $P_2$. PDTTs and DDTTs are equivalent to the notion of weak and strong tests as defined in [Boroday et al., 2007]: like PDTs and DDTs, a PDTs is a sequence that may reveal a difference between two hypotheses, whereas a DDTs is a sequence that will necessarily do so. In the case of deterministic plants, PDTs and DDTs coincide. Again, DDTs are symmetric: a DDT to discriminate $P_1$ from $P_2$ is also a DDT to discriminate $P_2$ from $P_1$.

For example, Fig. 2 shows two plants $P_1$ and $P_2$ with $I = \{L, H\}$ and $O = \{0, 1\}$. The input sequence $\sigma = L.L$ is a PDTs for $P_2$, $P_1$, because, for example, 0,1,0 is a possible output sequence of $P_2$ but not of $P_1$. The sequence $\sigma' = H.H$ is a DDTs for $P_2$, $P_1$, because the only possible output sequence 0,0,0 cannot be produced by $P_1$.

### 4.1 Characterizing PDTTs and DDTTs

We give QCSP formulas that encode the problem of finding PDTTs and DDTTs with a length at least or equal to $k$. Using QCSPs, feasible traces of length $k$ of a plant can be captured as follows: a sequence of inputs and outputs is feasible, if there exists a sequence of states such that for any two consecutive states $x, x'$ along the sequence, the respective input $i$ and output $o$ must be consistent with the transition relation $\delta$ and the observation relation $\lambda$:

$$
\phi(i_1, \ldots, i_k, o_0, \ldots, o_k, X, \delta, \lambda) \equiv \exists x_0, \ldots, x_k \forall x, x', i, o \cdot \\
(\forall_{j=0}^{k}(x = x_j) \land (x' = x_{j+1}) \land \delta(x, i, x')) \land ((\forall_{j=0}^{k}(x = x_j) \land \delta(o = o_j) \rightarrow \lambda(x, o))
$$

From this, we can construct a QCSP formula that encodes the problem of finding a PDTs with a maximum path length of $k$:

$$
\exists i_1, \ldots, i_k \exists o_0, \ldots, o_k \cdot \phi(i_1, \ldots, i_k, o_0, \ldots, o_k, X, \delta, \lambda) \land \neg \phi(i_1, \ldots, i_k, o_0, \ldots, o_k, Y, \eta, \mu)
$$

(3)

Extending on (3), the following QCSP formula captures DDTs with a maximum path length of $k$:

$$
\exists i_1, \ldots, i_k \forall o_0, \ldots, o_k \cdot \phi(i_1, \ldots, i_k, o_0, \ldots, o_k, X, \delta, \lambda) \Rightarrow \neg \phi(i_1, \ldots, i_k, o_0, \ldots, o_k, Y, \eta, \mu)
$$

(4)

We compare this to the approach in [Esser and Struss, 2007], which is based on unrolling automata into a constraint network using $k$ copies of the transition relation and the observation relation, and then applying test methods for logical models as discussed in Sec. 3. The advantage of the QCSP-based encoding (3,4) is that for any $k$, it requires only a single copy of the transition relation and the observation relation, which are the biggest components in most automata model specifications. Thus, the size of the formula will grow much more moderately with the number of time steps $k$ than the constraint network in [Esser and Struss, 2007]. However, it is still open to what extent current QCSP/QBF solvers can exploit this more compact encoding of the test generation problem, and turn it into actual performance improvements (see also Sec. 6).

As for the complexity, for non-deterministic finite-state machines it has been shown that the problem of uniquely identifying its initial state from its input and output behavior is PSpace-complete [Alur et al., 1995]. This problem is equivalent to the problem of designing a sequence of inputs that allows to unambiguously distinguish among two non-deterministic finite-state machines (with known initial states), and therefore equivalent to the problem of finding DDTs:

**Proposition 2** The problem of finding DDTs is PSpace-complete.

To our knowledge, the complexity of finding PDTs is still unknown, but it is likely that this problem is also PSpace-complete.

### 5 Adaptive Testing

As discussed above, the QCSP (game-theoretic) framework is useful to compactly express, analyze and solve different variants of (known) model-based testing problems. However, in addition, it can also serve as a starting point to tackle new classes of problems that are closer to the practice of testing. Recall that in Def. 3, tests are assumed to consist of (complete) assignments to controllable variables $I$. Actually, looking closely, there are two assumptions underlying this definition, namely that i) testing is performed as a two-step process where one first sets the inputs and then observes the outputs, and ii) the controllable variables characterize all relevant causal inputs to the system. In the following, we seek to relax these two assumptions.

Relaxing the first assumption means to extend testing from the problem of finding input assignments to the problem of finding adaptive tests, where input variables can be set depending on the values of observed output variables. Such an adaptive sequence is in fact a strategy that describes which values the input variables must be given in response to the values of observed variables (represented, for example, as a decision tree). Generating such adaptive strategies goes beyond the theory in [Struss, 1994], which assumed that tests consist of assignments (patterns) for the input variables, but it is possible in the QCSP framework. For logical models, adaptive tests can be captured using the following modified QCSP formula (assuming, without loss of generality, that the number of input variables equals the number of output variables):

$$
\exists i_1 \forall o_1 \ldots \exists i_k \forall o_k \forall u_1 \ldots u_k \cdot M_i \rightarrow \neg M_j
$$

(5)

While the non-adaptive version of DDTs (Sec. 3.1) is $\Sigma_2^P$-complete, the adaptive version (5) is harder to compute.
(PSpace-complete). For the case of (non-deterministic) automata models, we get a similar picture: it has been shown [Alur et al., 1995] that finding such adaptive distinguishing sequences is in ExpTime, and therefore even harder than the problem of finding DDTs. Surprisingly, for deterministic automata models, the problem is polynomial and therefore easier than the DDTs problem [Lee and Yannakakis, 1994]. For model-based testing, this leads to two interesting insights: first, since the class of adaptive tests (observation-dependent inputs) generalizes the class of non-adaptive tests (observation-independent inputs), from the two classes being different it follows that both for logical (constraint-based) models and for automata models, adaptive tests are strictly more powerful in the sense that an adaptive test might exist even if a non-adaptive test does not exist. Second, the more general form of adaptive (observation-dependent) testing is not just more powerful, but for deterministic (or nearly deterministic) models it is even computationally preferable over non-adaptive testing.

As already noted in the introduction, relaxing the second assumption (controllable variables characterize all relevant causal inputs to the system) is often a practical necessity: during testing, even in a highly controlled environment such as an automotive test-bed, there might be variables or parameters that influence the system’s behavior, but whose values cannot be completely controlled. For logical models, this scenario of testing under limited controllability can be captured using a modification of (2). Let \( I \) be partitioned into input variables \( I_s = \{i_1, \ldots, i_n\} \) that can be controlled (set during testing), and input variables \( I_{nc} = \{i_{n+1}, \ldots, i_m\} \) that can be observed but not controlled. Then a definitely discriminating test exists iff the following formula is satisfiable:

\[
\forall i_{n+1} \ldots i_m \exists i_1 \ldots i_n \forall o_1 \ldots o_m \forall u_1 \ldots u_k : M_1 \rightarrow \neg M_2
\] (6)

Again, this problem is strictly harder than the DDT problem (Sec. 3.1). Also note again that while solutions to (1) and (2) are simply assignments to the values of the input variables, solutions to (6) are in general more complex and correspond to a strategy or policy that states how the values of the controllable variables \( I_s \) must be set depending on the values of the non-controllable variables \( I_{nc} \). To illustrate this, consider again the example in Fig. 1, but assume that variable \( x \) can’t be controlled. According to Def. 3, no DDT exists in this case, as the possible observations for \( v \) will always overlap for the two hypotheses \( M_1 \) and \( M_2 \). However, there exists a test strategy to distinguish \( M_1 \) from \( M_2 \), which consists of setting \( y \) depending on the value of \( x \): choose input \( y \leftarrow H \), \( z \leftarrow H \) if \( x = L \), and choose input \( y \leftarrow L \), \( z \leftarrow H \) if \( x = H \). Again, generating test for such systems with limited controllability goes beyond the theory in [Struss, 1994], but it is possible in the QCSP framework.

We are currently working on merging the two sources of non-determinism in testing (non-deterministic behavior of the system and limited controllability of the system) into one common framework for QCSP-based adaptive testing.

6 Prototypic Implementation of QCSP-based Testing

We have conducted preliminary experiments of QCSP-based test generation with the solvers Qecode [Benedetti et al., 2007] and sKizzo [Benedetti, 2005] (since the present version of Qecode does not allow one to extract solutions from satisfiable instances, we transform the instance into QBF and use sKizzo to extract solutions). So far, we have implemented several examples of non-adaptive and adaptive test generation for logical models (Sec. 3), and a small example of non-adaptive test generation for automata models (Sec. 4). At the moment, these examples are still too small for a meaningful performance comparison of our approach (in the non-adaptive case) to the approach in [Struss, 1994; Esser and Struss, 2007]. However, within the CoTeSys (cognition in technical systems) project [Beetz et al., 2007], the QCSP-based testing prototype will be evaluated using larger models of an intelligent factory test-bed.

Figure 3 shows solutions generated from equations (2) and (6) for the example in Fig. 1. The solutions are represented in the form of BDDs with complemented arcs (see [Benedetti, 2005]), where \( \neg x \) stands for \( x = L \), \( x \) stands for \( x = H \), etc. The lefthand side of the figure shows the strategy (in this case, a simple set of assignments) that is generated if variables \( x, y, z \) are specified as controllable (input) variables, whereas the righthand side of the figure shows the strategy when only \( y, z \) are controllable (in this case, \( y \) must be set depending on the value of \( x \)). No solution (definitely discriminating test strategy for the fault) is found if only \( z \) is assumed to be controllable.

7 Discussion and Future Work

We reviewed an existing theory [Struss, 1994] of diagnostic testing for physical systems, which defines a weaker (PDTs) and a stronger form (DDTs) of test inputs, and showed how it can be framed as QCSP solving. For the first time, we give precise results on the complexity of this problem (in between NP and PSpace). Furthermore, we showed how assumptions in this theory about the complete controllability of system inputs can be relaxed and lead to a strictly more powerful class of tests, where inputs are intelligent set in reaction to observed values. Such test strategies go beyond the test pattern approach of the existing theory, but they can be captured in the QCSP framework. We also extended the QCSP-based formulation of testing to the case of plants modeled as non-deterministic automata.
While there exist approaches that solve non-deterministic testing problems using classic constraint solvers [Esser and Struss, 2007] and model checkers [Boroday et al., 2007], we believe that the QCSP-based representation can be advantageous for several reasons:

- First, as noted in Sec. 4, the QCSP encoding is quite compact. While it is not yet clear if this theoretical advantage can indeed be capitalized by current solver technology, there are at least hints [Benedetti and Mangassarian, 2008] that it can lead to performance improvements as more sophisticated techniques are added to these solvers.

- Second, because QCSPs are kind of a natural generalization of CSPs, it is not too difficult to lift extensions of CSPs such as soft constraints and optimization to QCSPs. In fact, the next release of the QCSP solver we used for our experiments (Qecode) contains optimization extensions. Thus, using the QCSP-based formulation, it will be relatively easy to extend model-based testing in order to generate, for instance, cost-optimal test strategies or probabilistic test strategies that most likely discriminate fault hypotheses.

- Third, existing methods to combat search space complexity by automated abstraction of constraints can be straightforwardly extended from CSPs to QCSPs and thus be adapted to the context of model-based testing with limited effort. Based on our previous work in this direction [Sachenbacher and Struss, 2005; Maier and Sachenbacher, 2008] and related work in [Clarke et al., 2003], we plan to devise an abstraction-refinement method for constraint-based testing of hybrid systems.

We are also currently working on larger, more realistic examples to evaluate our QCSP-based testing approach. In particular, in the future we seek to complement passive verification tools [Cimatti et al., 2003] for embedded autonomous controllers [Williams et al., 2003] with a capability to generate test strategies that can actively reveal faults.

Acknowledgments

The authors would like to thank Michael Esser, Paul Maier, and Peter Struss for useful comments.

References


A Jointree Algorithm for Diagnosability and its Application to the Verification of Distributed Software Systems∗

Anika Schumann

The Australian National University
Canberra ACT 0200, Australia
anika.schumann@anu.edu.au
tel: +61 2 6267 6216
fax: +61 2 6125 8651

Wolfgang Mayer

University of South Australia
Adelaide SA 5095, Australia
{mayer,mst}@cs.unisa.edu.au
tel: +61 8 8302 3965
fax: +61 8 8302 3988

Markus Stumptner

University of South Australia
Adelaide SA 5095, Australia
{mayer,mst}@cs.unisa.edu.au
tel: +61 8 8302 3965
fax: +61 8 8302 3988

Abstract

Diagnosability is an essential property that determines how accurate any diagnostic reasoning can be on a system. While diagnosability in a discrete event system can be decided by synchronising finite state machines representing ambiguous paths in individual subsystems, this synchronisation operation remains prohibitively complex.

We propose a novel algorithm that exploits structure and locality properties of a system to avoid expensive synchronisation operations. By propagating concise summary information reflecting diagnosability of small subsystems, diagnosability of the entire system is computed incrementally. As a result, we obtain an efficient algorithm that can not only decide (non)diagnosability but that is also applicable in scenarios where computational resources are limited. We also show how our algorithm can be applied to analyse distributed (software) systems.

1 Introduction

Automated fault diagnosis has significant practical impact by improving reliability and facilitating maintenance of systems. Given a monitor continuously receiving observations from a dynamic event-driven system, diagnostic algorithms detect possible fault events that explain the observations. For many applications, it is not sufficient to identify what faults could have occurred; rather one wishes to know what faults have definitely occurred. Computing the latter in general requires diagnosability of the system, that is, the guarantee that the occurrence of a fault can be detected with certainty after a finite number of subsequent observations [Sampath et al., 1995]. Consequently, diagnosability analysis of the system should be performed before any diagnostic reasoning. The diagnosability results then help in choosing the type of diagnostic algorithm that can be performed and provide some information of how to change the system to make it more diagnosable.

While diagnosability of existing systems has been actively researched, the issue has become relevant also in the design and development phase of new systems, where diagnosability is used to verify that faults can be detected and isolated easily. Given that considerable parts of modern complex systems are implemented in software, it has become desirable to apply similar techniques also to computer programs. In this context, the sensor placement problem to ensure diagnosability translates into the debugging problem, where specific faults in executable programs must be distinguished and isolated. While programs are generally more accessible than integrated physical systems, tight constraints on developer effort and limited computational resources on embedded devices in distributed settings do restrict monitoring and diagnostic solutions. Hence, establishing sufficient but cost-effective fault isolation frameworks in software design remains a vital issue that must be addressed as part of the overall system design phase. In particular, systems should be designed such that likely faults in software can be isolated quickly and unambiguously. Fortunately, the same analysis techniques can help in the design of both hard- and software systems: in physical systems, diagnosability amounts to deciding the sensor placement problem, while in the software domain possible probes and monitoring frameworks must be implemented at strategic interfaces between dependent subsystems. In the following presentation, both domains are used interchangeably.

In this paper, we propose a formal framework for checking diagnosability of event-driven systems which is mainly motivated by two facts. Checking diagnosability means determining the existence of two behaviours in the system that are not distinguishable. However, in realistic systems, there is a combinatorial explosion of the search space that forbids the practical use of classical and centralised diagnosability checking methods [Sampath et al., 1995] like the twin plant method [Jiang et al., 2001; Yoo and LaFortune, 2002].

Our proposal makes several contributions to solving the diagnosability problem. The first one is the definition of a new theoretical framework where the classical diagnosability problem is described as a distributed search problem. Instead of searching for indistinguishable behaviours in a global model, we propose to distribute the search based on local twin plants [Pencolé, 2004], represented as finite state machines (FSMs). Specifically, we exploit modularity of a system by organising the system components into a tree structure, the jointree,
where each node of the tree is assigned a subset of the local twin plants whose collective set of events has a size bounded by the treewidth of the system. Once the jointree is constructed we need only synchronise the twin plants in each jointree node, and all further computation takes the form of message passing along the edges of the jointree. Using the jointree properties we show that after exchanging two messages per edge, the FSMs in the tree are collectively consistent. This allows to decide diagnosability by considering these FSMs in sequence instead of the large global twin plant.

We describe how messages represented as FSMs are computed based on projections of a FSM onto a subset of its events, and how diagnosability information can be propagated along with the messages. We employ an iterative procedure such that only a subset of the jointree is considered at a time, terminating the algorithm once a subset sufficient for deciding diagnosability has been determined. Our approach to use selective message passing in a jointree improves upon previous work by Pencolé (2004) in that we relax some of the assumptions underlying earlier work and achieve greater scalability by employing more efficient synchronisation mechanisms.

Since diagnosability analysis is a complex problem, our algorithm explicitly accounts for the possibility that the available resources may not be sufficient to calculate the precise solution for a given problem. Our incremental analysis algorithm ensures that in such cases it is able to provide an approximate solution to the diagnosability problem. Specifically, a sub-system where the existence of indistinguishable behaviours has been established but not yet verified against the rest of the system is returned. While an approximate solution cannot be used to verify that a system is definitely (non)diagnosable, it is useful to show that on-line monitoring of this particular subsystem will not be sufficient to detect occurrences of the fault.

This paper is organised as follows: in Section 2 we summarise the Twin Plant method of addressing the diagnosability problem for discrete event systems and outline the basic principles underlying jointrees. Section 3 presents our approach to use jointrees for diagnosability analysis, discussing our message-passing scheme and iterative algorithm. Differences to related work are discussed in Section 4, followed by a summary of our contributions and possible future research directions.

2 Background

In this section we review the definition of diagnosability and the twin plant approach to diagnosability checking, and give a short introduction to jointrees.

2.1 Diagnosability of Discrete Event Systems

Similar to the diagnosis of discrete-event systems we consider a system of distributed systems $G_1, \ldots, G_n$. The behaviour of each component can be represented as a finite state machine (FSM) $G_i = (X_i, \Sigma_i, x_{0i}, T_i)$ where $X_i$ is the set of states, $\Sigma_i$ is the set of events, $x_{0i}$ is the initial state, and $T_i$ is the transition relation ($T_i \subseteq X_i \times \Sigma_i \times X_i$). The set of events $\Sigma_i$ is divided into four disjoint subsets: observable events $\Sigma_{ob}$, communication events $\Sigma_{com}$, shared with other components, unobservable fault events $\Sigma_{fb}$, and other unobservable events $\Sigma_{un}$. Without loss of generality the communication events are assumed to be unobservable and observable and fault events to be specific to a $G_i$.

**Example 1** Assume a system of communicating processes is to be analysed to assess whether sufficient monitoring and logging capabilities have been put in place to detect particular types of faults in the system.

In our model, each process provides particular services to its peers and uses services provided by others by means of exchanging messages. We abstract from concrete data contained in a message and represent each type of message as event. Hence, events and messages serve to coordinate the overall execution of processes in our system.

Figure 1 depicts a small distributed system of three subsystems represented as communicating FSMs: process $G_S$ represents a server process that, once initialised, receives and executes requests from clients and returns the result. Subsystem $G_C$ represents a client that communicates with the server process by sending requests and processing replies. Subsystem $G_L$ implements a message archive where alerts are logged and stored for later analysis.

Interactions between subsystems are modelled as shared events; $G_S$ and $G_C$ communicate via events request and reply, while $G_S$ sends alerts to $G_L$ using an alert message.

Observe events in our model correspond to activity that can be perceived by a user’s or administrator’s point of view. In $G_C$, the completion of the initial setup stage, where parameters for the subsequent processing cycle are set, and the result of each request are directly visible. On the server side, event ready (denoting the completion of the initialisation stage) can be observed by the operator and a log file with entries for each request (event log) can be inspected. In $G_L$, the addition of a newly arrived message can be observed via event write.

In $G_S$, the initialisation step init cannot be observed directly. We further assume that the initialisation can fail (event fail); in this case, an alert event alert is sent to the logging subsystem. We assume that the system continues to execute, but some requests may not complete successfully due to failed initialisation of $G_S$.

We use this example to show how diagnosability analysis can help to assess whether the observable events are sufficient to infer the presence or absence of event fail.

---


*1The jointree algorithm applied to a short example of interacting physical components is in [Schumann and Huang, 2008]."
While the model in Figure 1 seems intuitively correct, we will show that the observable events are insufficient to decide the presence or absence of event fail. While the alert message is designed to make a failure observable in the logging component, \( G_L \) may delay its write operation indefinitely. Hence, this system becomes nondiagnosable. Results like these can be useful for system design, where requirements for different subsystems are laid out. By analysing a proposed design with respect to diagnosability of faults, requirements on logging and monitoring facilities can be verified.

Note that a monolithic model for the entire system is implicitly defined as the synchronised product, \( G = \text{Sync}(G_1, \ldots, G_n)^2 \) of all component models.

A fault \( F \in \bigcup \Sigma_i \) of the system is diagnosable iff its (unobservable) occurrence can always be deduced after finite delay [Sampath et al., 1995]. In other words, a fault is not diagnosable if there exist two infinite paths from the initial state which contain the same infinite sequence of observable events, but only one sequence contains a fault.

More formally, let \( pF \) denote a path starting from the initial state of the system and ending with the occurrence of a fault \( F \) in a state \( xF \), let \( SF \) denote a finite path starting from \( xF \), and let \( \text{obs}(p) \) denote the sequence of observable events in a path \( p \). As in [Sampath et al., 1995], we assume that (i) the system is live (there is a transition from every state), and (ii) the observable behaviour of the system is live (\( \text{obs}(p) \) is infinite for each infinite path \( p \) of the system). Then, diagnosability of a system with respect to \( F \) can be defined as follows:

**Definition 1 (Diagnosability)**: \( F \) is diagnosable iff

\[
\exists d \in \mathbb{N}, \forall pF \in SF, |\text{obs}(SF)| > d \Rightarrow (\forall p, \text{obs}(p) = \text{obs}(pF) \Rightarrow F \text{ occurs in } p).
\]

Diagnosability checking thus requires the search for two infinite cyclic paths \( p \) and \( p' \) where \( F \) occurs in \( p \) but not in \( p' \), such that \( \text{obs}(p) = \text{obs}(p') \). The pair \((p, p')\) is called a critical pair [Cimatti, Pecheur, & Cavada, 2003]. Unless stated otherwise, we will use \text{path} to refer to a path that starts from the initial state of the system.

### 2.2 Twin plant for diagnosability checking

The idea of the twin plant is to build a FSM that compares representing individual components, FSMs representing larger subsystems are laid out. By analysing a proposed design with respect to diagnosability, requirements on logging and monitoring facilities can be verified.

An interactive diagnoser (bottom label) and one in the right subsequent figures. State labels are composed of a state in the left interactive diagnoser (middle label) and one in the right interactive diagnoser (bottom label). Oval nodes represent fail-nondiagnosable states. In this example, a part of the twin plans is shown where the fault cannot be deduced in all but the initial state.

A fault \( F \) is diagnosable in system \( G \) iff its global twin plant (GTP) \( \text{Sync}(G_1, \ldots, G_n) \) has no path \( p \) with a cycle containing at least one observable event and one \( F \)-nondiagnosable state [Schumann and Pencolé, 2007]. Such a path \( p \) represents a critical pair \((p_1, p_2)\), and is called a critical path. The oval nodes in Figure 2, for example, form part of a critical path.

The twin plant method searches for such a path in the GTP. However, the GTP may be prohibitively large to perform this global analysis. In this paper, we propose a new algorithm that avoids building the global twin plant and operates on local twin plants instead. Since the existence of a critical path in a local twin plant does not imply nondiagnosability of the global system, the results of the local analysis must be propagated to other twin plants to decide diagnosability. As our main contribution in this paper we present a novel algorithm that ex-
exploits a jointree to efficiently perform propagation and global diagnosability assessment.

### 2.3 Jointrees

Jointrees have been a classical tool in probabilistic reasoning and constraint processing [Shenoy and Shafer, 1986; Dechter, 2003], and correspond to tree decompositions known in graph theory [Robertson and Seymour, 1986]. For our purposes, a jointree is a tree whose nodes are labelled with sets of events satisfying two conditions:

**Definition 2 (Jointree)** Given a set of FSMs $G_1, \ldots, G_n$ defined over events $\Sigma_1, \ldots, \Sigma_n$ respectively, a jointree is a directed tree where each node is labelled with a subset of $\Sigma = \bigcup_i \Sigma_i$ such that

- every $\Sigma_i$ is contained in at least one node, and
- if an event is shared by two distinct nodes, then it also occurs in every node on the path connecting the nodes.

**Example 4** Figure 3 (left) depicts a jointree for our three processes described in Figure 1. The label on each edge represents the intersection of the two neighbouring nodes, known as the separator.

Once a jointree has been constructed, each FSM $G_i$ is assigned to a node that contains its events $\Sigma_i$. Figure 3 (right) depicts such an assignment. Note that in general each node may be assigned multiple FSMs.

Once the FSMs have been assigned, the structure of the jointree can be exploited to guide our diagnosability assessment. In the following sections we show that it is sufficient to synchronise all FSMs assigned to the same node in the tree, followed by two message passing phases. The properties of the jointree then guarantee that consistency among all the FSMs has been achieved.

The efficiency of jointree propagation depends on the size of the FSMs computed. Hence it is desirable to minimise the number of FSMs assigned to a single node. For this purpose we can also use the well-known heuristics that minimise the number of events labelling a jointree node. The size of the largest label, minus 1, is known as the width of the jointree, and the minimum width among all possible jointrees for a given system is known as the treewidth of the system [Robertson and Seymour, 1986]. Efficient polynomial-time procedures, such as the min-fill heuristic, can be used to create jointrees of low width by exploiting system structure [Dechter, 2003]. These procedures also guarantee that the jointree nodes are labelled in such a way that no FSM is assigned to more than one node. Note further that nodes might be labelled with events that do not occur in any of the FSMs assigned to it. This is the case if FSMs interact in a cyclic way.

### 3 A Jointree Algorithm for Diagnosability

The synchronisation of all twin plants in a jointree would solve the diagnosability problem. However, for large systems this can be prohibitively expensive. This complexity can be avoided by synchronising only the twin plants in each jointree node, followed by message passing between adjacent tree nodes to propagate local results to a wider scope. After two cycles of message passing and synchronisation with local FSMs, global diagnosability can be decided.

Jointrees admit a generic message passing method that achieves consistency among the nodes [Dechter, 2003]. In our case this translates into a method that achieves consistency of all FSMs labelling the jointree nodes. Here, the messages will themselves be FSMs. In the following we present an algorithm to compute these messages and show how diagnosability information can be propagated correctly between nodes. Subsequently, we detail our iterative algorithm that addresses the diagnosability problem.
3.1 Establishing consistency

While FSMs assigned to the same tree node are synchronised directly to obtain a local picture of the system behaviour, messages must be exchanged to achieve consistency between nodes.

Definition 3 (Global Consistency; Completeness) A FSM \( G_i \) with events \( \Sigma_i \) is globally consistent with respect to FSMs \( G_1, \ldots, G_n \) iff for every path \( p_i \) in \( G_i \) there exists a path \( p \) in the synchronised product \( \text{Sync}(G_1, \ldots, G_n) \) that has with respect to \( \Sigma_i \) the same event sequence as \( p_i \). A FSM \( G_i \) is complete iff it contains all globally consistent paths of \( G_i \).

Each edge in a jointree partitions the tree into two subtrees, and a message sent over an edge represents a summary of the collective behaviour permitted by the sending side of the partition. A major advantage of this method is that this summary needs only to mention events given by the separator labelling the edge; the jointree construction ensures that this equals the intersection of the two sets of events across the partition.

A message can be computed by projecting a FSM onto a subset of its events.

Definition 4 (Projection) The projection \( \Pi_{\Sigma_i}(G) = (X', \Sigma', x_0, T') \) of a FSM \( G \) on events \( \Sigma' \subseteq \Sigma \) is obtained from \( G \) by first contracting all transitions not labelled by an event in \( \Sigma' \) and then removing all states (except the initial state \( x_0 \)) that are not the target of any transition in the new set of transitions \( T' \). More formally, \( T' \) is given as follows:

\[
T' = \left\{ x \xrightarrow{\sigma'} x' \mid x, x' \in X' \text{ and } \sigma' \in \Sigma' \text{ and } \exists x \xrightarrow{\sigma_n} x_1 \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\sigma_{k+1}} x' \text{ in } G \text{ such that } \sigma_i \notin \Sigma' \forall i = 1, \ldots, k \right\}
\]

Figure 4 shows the result of projecting \( G_c \) on its shared events.

![Figure 4: Projection \( \Pi_{\{\text{request, reply}\}}(G_c) \)](image)

3.2 Message passing

To achieve consistency among the synchronised FSMs in a tree, each node requires a summary of the behaviour permitted by the FSMs in the remaining tree. Given the jointree properties, these summaries can be computed in only two passes over the jointree: one inward pass, in which the root "pulls" messages towards it from the rest of the tree, and one outward pass, in which the root "pushes" messages away from it towards the leaves. Once all messages have been exchanged, the FSM in each node is updated to reflect the information received from neighbouring nodes. As a result, all FSMs are complete and consistent.

The process starts by designating any node of the tree as root. Then, in the first, inward pass, beginning with the leaves each node sends a message to its (unique) neighbour \( n \) in the direction of the root. To compute this message, its FSM is synchronised with all messages it receives from its other neighbours (leaves do not have "other neighbours" and hence skip this step). The message that is subsequently sent to the node \( p \) closer to the tree root is the projection of this FSM onto the separator between \( n \) and \( p \).

In the second, outward pass, each node (except the root) receives a message from its (unique) neighbour in the direction of the root. Again, a message is computed by synchronising a node’s FSM with all messages it received from its other neighbours and by projecting the result onto the separator events between itself and the receiver of the message.

Finally, each node updates its associated FSM by synchronisation with messages received from its neighbours. As a result, each FSM \( G_i' \) represents exactly the behaviour that is complete and possible in the global model implicitly defined by the jointree.

Example 5 Figure 5 illustrates the inward and outward propagation steps performed on the jointree of Figure 3 (right).

![Figure 5: Inward (left) and outward (right) message propagation using jointrees, where \( \Sigma = \{l: \text{request}, r: \text{request}, l: \text{reply}, r: \text{reply}\} \) and \( \Sigma' = \{l: \text{alert}, r: \text{alert}\} \).](image)

We have shown that after message propagation has completed, the FSMs in each node are consistent and complete.

Theorem 1 Every FSM \( G_i' \) labelling a jointree node is complete and consistent with respect to all other FSMs \( G_1, \ldots, G_n \) of the tree once it has been synchronised with all received messages.

In particular it follows that for every path \( p_i \) in \( G_i' \) there is also an equivalent path \( p_i \) in \( G_i' \) defined over the same event sequence as \( p \), and vice versa.

While Theorem 1 establishes desirable properties, it can be shown that it is insufficient to decide diagnosability, since some critical paths may be lost due to the projection operation.

In general, we need to ensure that for every critical path \( p \) in \( G_i' \) there is also an equivalent critical path \( p_i \) in \( G_i' \). This requires the propagation of diagnosability information in addition to the message passing algorithm outlined previously.

3.3 Propagation of diagnosability information

In the rest of the section we will assume that (i) the twin plants representing subsystems have been assigned to appropriate jointree nodes and synchronised within each node, (ii) \( G_F \) is the FSM containing the fault F whose diagnosability is to be checked, and (iii) the node containing the twin plant \( G_F \) is chosen as root.
It has been shown that if any twin plant of a jointree node contains a consistent critical path, then the fault \( F \) is nondiagnosable [Schumann, 2007]. Hence, diagnosability can be decided by searching for critical paths in the individual FSMs in the jointree after the message passing phase is completed.

The root can already be examined for critical paths after the inward propagation phase: (i) the synchronisation of the root with all its incoming messages results in a globally consistent twin plant, and (ii), since the fault \( F \) appears in the root, the FSM already contains diagnosability information, that is, the classification of states into diagnosable and nondiagnosable ones. If the root does not contain a nondiagnosable state, the entire system is known to be diagnosable. Otherwise, the outward propagation phase must be carried out to determine whether another jointree node has a critical path.

Once propagation is complete, every state of a twin plant comprises a tuple \((x_1, \ldots, x_n)\). In particular, each state contains a state from \( G_r \) that has been received and synchronised with the local FSM as part of the messages pushed from the root in the outward propagation phase. To ensure diagnosability information is preserved, we must ensure that no path to a nondiagnosable state is lost in this process.

Recall that the projection operation applied to compute the outward message removes all states that are no longer a target of a transition labelled by a separator event in \( \Sigma \). This can lead to the removal of nondiagnosable states, resulting in the incomplete propagation of diagnosability information.

**Example 6** Consider the twin plant \( G_u \) shown in Figure 6 (left). Assume a message \( \mathcal{P}_u = \Pi_{\{s_1\}}(G_u) \) is to be computed with respect to event set \( \{s_1\} \) and sent to \( G_v \). By projecting \( G_u \) onto \( \{s_1\} \), the nondiagnosable state \( u_1 \) is eliminated. This results in the consistent twin plant \( \hat{G}_u = \text{Sync}(\Pi_{\{s_1\}}(G_u), G_v) \) obtained by synchronisation of \( \mathcal{P}_u \) with \( G_v \). However, \( \hat{G}_u \) does not contain any critical paths, although it should contain one (as shown by the properly synchronised FSM \( \hat{G}_w = \Pi_{\{s_1\}}(\text{Sync}(G_u, G_v)) \)).

![Figure 6: FSMs \( \hat{G}_u, \mathcal{P}_u, \hat{G}_v, \Sigma \), and \( \hat{G}_v \) (from left to right).](image)

We therefore need to ensure that every message passed on from \( \hat{G} \) to \( \hat{G}' \) via the separator events \( \Sigma_{sep} \) will lead to a consistent twin plant \( \hat{G}' \) that has a critical path iff \( \Pi_{\Sigma_{sep}}(\text{Sync}(\hat{G}, \hat{G}')) \) has one. This guarantees that \( \hat{G}' \) has a critical path iff \( \Pi_{\Sigma_{sep}}(\text{Sync}(\hat{G}, \hat{G}')) \) has one, where \( \Sigma_{sep} \) is the event set labelling the jointree node of \( \hat{G}' \).

To achieve this it is necessary to annotate every diagnosable state \( \hat{x} \) in a message to capture whether it has a nondiagnosable local future, that is, whether there is a transition sequence starting in \( \hat{x} \) and leading to a nondiagnosable state \( \hat{x}_t \) such that none of the transition events is kept in the projection.

**Definition 5 (Nondiagnosable Local Future)** Let \( \hat{G} \) and \( \hat{G}' \) be two FSMs associated with adjacent nodes in a jointree connected by an edge labelled \( \Sigma_{sep} \), and let \( \hat{x}_t \) denote a nondiagnosable state in \( \hat{G} \). Then, a diagnosable state \( \hat{x} \in \hat{G} \) has a nondiagnosable local future iff there exists a transition sequence

\[
\hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k
\]

in \( \hat{G} \) such that none of the events \( \sigma_1, \ldots, \sigma_k \) are in \( \Sigma_{sep} \).

We capture this information by adding additional nondiagnosable subgraphs to the FSM \( \Pi_{\Sigma_{sep}}(\hat{G}) \) obtained by projection of \( \hat{G} \): for every diagnosable state \( \hat{x} \in \hat{G} \) that has a nondiagnosable local future w.r.t. \( \Sigma_{sep} \), a nondiagnosable extended terminal state \( \text{ext}(\hat{x}) \) and a transition \( \hat{x} \xrightarrow{\text{ext}} \text{ext}(\hat{x}) \) are added to ensure that the critical path is not lost in the projection.

**Example 7** The left part of Figure 7 illustrates the message \( M_{SL} \) sent from \( G_S \) (see Figure 2) to \( G_L \). The only diagnosable state \( x_0 \) in \( G_S \) does not have a nondiagnosable local future, since all outgoing transitions are kept in the projection \( \Pi_{\{\text{alert}\}} \), and the nondiagnosable state \( x_1 \) is included in \( M_{SL} \).

In contrast, the state \( x_0 \) of the message \( M_{SC} \) shown on the right of Figure 7 does have a nondiagnosable local future w.r.t. \( \{l : \text{request}, r : \text{request}, l : \text{reply}, r : \text{reply}\} \) according to Definition 5. The path \( x_0 \xrightarrow{l: \text{alert}} x_1 \) in \( G_S \) leads to the nondiagnosable state \( x_1 \), but is eliminated by the projection \( \Pi_{\{\text{request,reply}\}} \). Hence, an extended state \( \text{ext}(x_0) \) must be introduced in \( M_{SC} \) as depicted on the right of Figure 7.

![Figure 7: Message \( M_{SL} \) (left) and part of message \( M_{SC} \) (right). Circles denote nondiagnosable states and hexagon shapes extended states, respectively.](image)

Note that there is no need to introduce artificial states for a nondiagnosable state \( \hat{x}' \). This results from the fact that all states reachable from \( \hat{x}' \) via transitions labelled by events not kept in the projection can only be part of a nondiagnosable cycle if there is also a nondiagnosable cycle with state \( \hat{x}' \) (according to the synchronisation operation). Hence nondiagnosability can be verified correctly based only on the latter.

Using the extended messages diagnosability can be decided:

**Theorem 2** Fault \( F \) is diagnosable in \( G \) iff after both passes of jointree propagation with diagnosability information, no FSM in a jointree node has a critical path.

### 3.4 Iterative jointree propagation

Since the complexity of our approach results from the complexity of the message propagation (and not the jointree construction), the efficiency of the algorithm can further be improved by limiting the scope of propagation within the jointree. We propose an algorithm that iteratively extends propagation in the tree until a subtree sufficiently large to decide (non)diagnosability has been processed, or computational resources have been exhausted. In the latter case, our algorithm...
Algorithm 1 \textsc{CheckDiagnosability}(jointree: $J$)

1: $\hat{G} \leftarrow \emptyset$ \hspace{1cm} nodes in $J$ being considered
2: $\Sigma_{\text{int}} \leftarrow \emptyset$ \hspace{1cm} events internal to $\hat{G}$
3: repeat
4: \hspace{1cm} $v \leftarrow \text{PickNode}(J, \hat{G})$
5: \hspace{1cm} $(\hat{G}, \Sigma_{\text{int}}) \leftarrow \text{AddToScope}(v)$
6: \hspace{1cm} $\text{Propagate}(\hat{G})$
7: \hspace{1cm} $\hat{G}_{\Sigma_{\text{int}}} \leftarrow \text{ProjectPaths}(\hat{G}, \Sigma_{\text{int}})$
8: \hspace{1cm} $\text{Propagate}(\hat{G}_{\Sigma_{\text{int}}})$
9: \hspace{1cm} if $\text{ExistsTwinPlantWithCritPath}(\hat{G}_{\Sigma_{\text{int}}})$ then
10: \hspace{1.5cm} return $\text{GetCritPath}(\hat{G}_{\Sigma_{\text{int}}})$
11: \hspace{1cm} end if
12: until $\hat{G} = J$ or $\text{IsDiagnosable}(\text{root})$
13: if $\text{SufficientMemory}(\hat{G})$ then
14: \hspace{1cm} return "F is diagnosable"
15: else
16: \hspace{1cm} $\omega \leftarrow \text{set of components included in } \hat{G}$
17: \hspace{1cm} if $\text{ExistsTwinPlantWithCritPath}(\hat{G})$ then
18: \hspace{2cm} return "$\omega$ has a critical path"
19: \hspace{1cm} else
20: \hspace{2cm} return "$\omega$ has no critical path"
21: \hspace{1cm} end if
22: end if

returns a conservative approximation of the exact solution that can help to guide further analysis of a system.

The idea is that any critical path $p$ in the global twin plant can be detected by looking only at those twin plants that define events in $p$, since other twin plants cannot affect the path. Our aim is to find a critical path defined over as few events as possible to limit the scope of the jointree that must be processed and to lower computational effort.

We search for such a path by iteratively increasing the set of jointree nodes (twin plants) $\hat{G}$ under consideration and limit our search to critical paths defined over internal events $\Sigma_{\text{int}}$ in $\hat{G}$ that do not appear in the remaining jointree. If such a path can be found, nondiagnosability has been shown and the search terminates.

Algorithm 1 outlines our search procedure. Assume that $\text{root}$ denotes the root node chosen for propagation; function $\text{PickNode}$ returns $\text{root}$ upon initial invocation, and a node in $J$ neighbouring $\hat{G}$ on subsequent invocations. Node selection heuristics are discussed in the next section. Initially, the root node is the only source of diagnosability information. In each iteration a new node that has a neighbour in $\hat{G}$ is selected (line 4), expanding the scope of our search by adding it to $\hat{G}$ and updating event set $\Sigma_{\text{int}}$. Jointree propagation is then run twice:

1. On $\hat{G}$ (line 6) to remove inconsistent paths. This can lead to the removal of nondiagnosable states which in turn may cause the root to become diagnosable ($\text{IsDiagnosable}(\text{root})$ is true) and thus verify diagnosability;
2. On $\hat{G}_{\Sigma_{\text{int}}}$ (line 8), which is obtained by removing from all twin plants in $\hat{G}$ the transitions labelled by events not in $\Sigma_{\text{int}}$ (line 7). This allows to detect if a twin plant $\hat{G} \in \hat{G}$ has a critical path whose global consistency can be verified by considering only the twin plants in $\hat{G}$, since it does not contain any event that appears in the rest of the tree. In this case, Algorithm 1 stops and returns the critical path that implies nondiagnosability (line 10).

The algorithm continues until one of the following conditions is satisfied:

- The root node (and hence the entire system) has been shown diagnosable. Note that it is indeed sufficient to check only the root node, since if the root has no nondiagnosable states, none of the messages it propagates and hence no twin plant includes a nondiagnosable state.
- The entire jointree is considered, but none of the twin plants contains a critical path; hence the diagnosability of the system is verified (line 14). The case where root contains a critical path is covered by line 10, since $\hat{G} = J$ implies that $\Sigma_{\text{int}}$ contains all events.
- The available resources have been exhausted (lines 16–21). In this case the maximal subsystem $\omega$ for which the existence of critical paths has been decided (but not yet verified against the rest of the system) is returned.

Any critical path in $\omega$ can be interpreted as hint indicating nondiagnosability (of at least the isolated subsystem considered so far). In case critical paths exist in $\omega$, then the larger this subsystem is, naturally, the more likely the whole system is not diagnosable; otherwise the reverse is true. Such an approximate solution is also useful in that it implies that on-line monitoring of this particular subsystem will not be sufficient to reliably detect faults.

\textbf{Example 8} Applied to our running example, Algorithm 1 selects the jointree node containing $G_S$ in the initial iteration. Since the events in any critical path in $\hat{G}_S$ are not internal to $G_S$, neither diagnosability nor nondiagnosability can be established, and the scope of the search must be expanded to include another subsystem. Assume $G_L$ is selected and added to the scope, leading to $\Sigma_{\text{int}} = \{l: alert, r: alert, log, ready, write\}$. Again, (non)diagnosability cannot be decided since all critical paths in the $\hat{G}$ contain either a request or a reply event shared with $G_C$. After extending $\hat{G}$ with the remaining subsystem $G_C$, $\Sigma_{\text{int}}$ contains the events shared by $G_S$ and $G_C$. Now, a critical path in $\hat{G}_{\Sigma_{\text{int}}}$ exists and the algorithm terminates in line 10.

Had the size of the system exceeded the available resources after the second iteration, our algorithm would have returned that the subsystem $\{G_L, G_S\}$ is potentially nondiagnosable (line 18), approximating the exact result.

\textbf{3.5 Node selection heuristics}

The heuristics used to select a jointree node to explore next can have considerable impact on the number of nodes necessary to decide diagnosability. Instead of directly choosing a node, we select nodes based on the set of events that are introduced into $\Sigma_{\text{int}}$ by a candidate node.

Let $\Sigma_p$ denote the set of shared events appearing in a critical path $p$ in a twin plant $\hat{G} \in \hat{G}$. Our heuristic is to expand $\Sigma_{\text{int}}$
with a new event in $\Sigma_p \setminus \Sigma_{int}$ in the hope that $p$ may at some point evolve into a new critical path that contains only internal events. To further focus the search, we only consider events in $\Sigma_p \setminus \Sigma_{int}$ for paths $p$ for which $|\Sigma_p \setminus \Sigma_{int}|$ is minimal. Among these eligible events, we select one that appears in the fewest nodes outside $\mathcal{G}$. The idea here is to minimise the number of nodes that need to be included in $\mathcal{G}$ for that event to be internal. After choosing an event, we iteratively add to $\mathcal{G}$ the neighbouring nodes containing that event.

4 Related Work

The diagnosability problem of discrete-event systems was first addressed in [Sampath et al., 1995] by constructing a deterministic diagnoser for the global system model. The main drawback of this method is its space complexity that is exponential in the number of system states.

Jiang et al. (2001) and Yoo and Lafortune (2002) proposed different algorithms that are of polynomial complexity and introduce the twin plant method. The question of efficiency is also raised in [Cimatti, Pecheur, & Cavada, 2003] where the authors propose to use symbolic techniques to test a restrictive diagnosability property by taking advantage of efficient model-checking tools. However, diagnosability assessment remains exponential in the number of components, even when encoded by means of binary decision diagrams as in [Cimatti, Pecheur, & Cavada, 2003].

More recent work aims at establishing either diagnosability or nondiagnosability, but not both. The work by Rintanen and Grastien (2007) shows how to detect nondiagnosability by searching for critical paths using SAT. If the algorithm cannot find a critical path it does not imply that there is indeed none, and it remains unknown whether the system is diagnosable or not. Conversely, the decentralised approach of Schumann and Pencolé (2007) can only establish diagnosability.

The approach of Pencolé (2004) is the closest to ours. It is based on the assumption that the observable behaviour of every component is live, which is more restrictive than our assumption (and that of [Sampath et al., 1995]), namely that the observable behaviour of the system (but not necessarily that of individual components) is required to be live. This restriction implies that it is sufficient to only search for a critical path in the twin plant $G^F$ containing the fault. In [Pencolé, 2004] this is done by iteratively synchronising $G^F$ with other local twin plants until diagnosability can be decided. In comparison to our approach this corresponds to the synchronisation of all twin plants $G$ considered at each iterative step of Algorithm 1. We do not require this synchronisation but achieve consistency by propagating messages with bounded event sets. If we adopted similar liveness restriction, it would be sufficient to search the jointree root for critical paths and skip outward propagation.

5 Conclusion and Future Work

We have presented a new approach to attack the diagnosability problem that addresses the fundamental complexity bottleneck of the classical twin plant method. By limiting our iterative analysis to a subsystem at a time, both the construction of the global system model as well as the synchronisation of local twin plants for entire subsystems can be avoided. Instead, local twin plants are made consistent by passing messages in a jointree representing clusters of related system components represented as finite state machines. Even with this improved algorithm, computational resources may be insufficient to find an exact solution and our algorithm returns an approximate solution that may guide further analysis. As part of future work we plan to extend our approach such that possible causes of nondiagnosability can be isolated and to explore ways to restore diagnosability.

References


Qualitative Modeling for Diagnosis of Machines Transporting Rigid Objects

Peter Struss, Axel Kather, Dominik Schneider, Tobias Voigt

Abstract

We present models of various elements of a plant that involves the transportation of lumped material. An application context is provided by a project on diagnosing disturbances in food packaging plants and, more specifically, bottling plants. While there exist models of flow of homogeneous matters, such as liquid material in a hydraulic system, based on simultaneous equations of Kirchhoff/Ohm type, in our project we need to cope with non-negligible transportation time of objects and capture phenomena like the tailback of units (if transportation is blocked) or the propagation of gaps in the flow of units. Because the application context requires compositionality of the model, i.e. local, context-free models of the individual transportation elements, we are also facing the problem that whether or not a single element produces an output flow (or accepts an input flow) cannot be determined solely by the model of this element, but only through modeling the interaction with the subsequent element, which may block the output (or the previous one not providing the input). This issue is addressed by modeling the potential of an existing flow distinctly from the actual occurrence of a flow, an idea which also can enhance models of continuous flow.

1. Introduction

Modeling the flow of some matter in a system is quite widespread in model-based systems, e.g. in model-based diagnosis of hydraulic or pneumatic systems. At least under certain simplifying assumptions, mathematical first principles models exist, and it appears to be straightforward to abstract them into adequate input to a model-based problem solver. Typically, such models assume that the flowing matter is continuous and homogeneous and does not have to be modeled as an object or its detailed structure. And they usually incorporate the analogies to Kirchhoff’s and Ohm’s Laws, which leads to simultaneous equations that imply instantaneous propagation of pressure and disregard time needed by the matter to be transported through the system. There are classes of application domains that involve a flow of objects through a plant and, hence, suggest the use of some flow model, but require dropping some of the simplifying assumptions mentioned. One instance of this class is given by food packaging plants, which are subject to a diagnosis project we are carrying out, and, more specifically, by bottling plants, which we will use as an example in this paper. Such plants involve streams of objects of different types, bottles, crates, and pallets being the most prominent ones. On the one hand, modeling the transportation of individual objects is prohibitive or useless. On other hand, the abovementioned flow models of a homogeneous matter fail to capture essential features, such as gaps in the flow or the creation of a tailback by some blockage and its propagation through the plant in finite time. Furthermore, an inflow and outflow of a single transportation element of a line cannot definitely be predicted by a local model of this element, because they depend also on the supply of the previous element and the intake capacity of the following one, resp. As a consequence, we had to develop a model that

• includes transportation times,
• covers interrupted flows,
• handles the exchange of flows between neighboring elements appropriately.

The paper focuses on presenting a base model addressing the requirements (section 3), its validation through simulation (section 4) and a qualitative diagnosis model obtained from it (section 5), but does not discuss the diagnosis engine. The following section presents an application context of this work, namely bottling plants.

2. An Application Domain: Bottling Plants

Food packaging at industrial scale is carried out in high output packaging lines consisting of specific machines and conveyors. There are different machines for specific packaging tasks, such as primary packaging of food or beverages (e. g. with foil packs, pouches, or containers), secondary packaging (boxes, multipacks, crates, etc.), and tertiary packaging (e. g. pallets or displays). Additionally, machines for de-palletizing and unpacking of returnable bottles, cleaning, inspection and sorting out improper objects may be involved. Plant constellations are configured using one machine of a specific type or several ones in parallel. Machines of different types are connected by conveyors. Because of the high speeds and output rates (up to 100.000 packages per hour), machines and
Conveyors are failure-sensitive with an availability degree of 92-98 percent.

As a specific example for packaging plants, our project considers bottling plants for beverages (e.g. the one shown in Fig 1).

In order to fill beverages into returnable bottles, the material flows of pallets, crates, and bottles (plus labels, glue, etc.) need to be coordinated. This leads to complex line configurations comprised of machines that remove crates from pallets and bottles from crates, process, inspect, or sort objects, and package different types of objects (Fig. 2 shows an abstract, but typical example).

To prevent oxygen intake or microbiological contaminations of the beverage, the filling process should not be interrupted. Therefore transportation by consecutive machines needs to be decoupled. Otherwise, each individual failure would inevitably cause downtime of the entire plant. In particular, this would stop the filling process and decrease the efficiency of the entire production. To prevent this, the conveyors of bottling plants are designed as transporting buffers like the abstract bottle conveyor shown in Fig. 3.

Transporting buffers perform two tasks. One is to carry the objects from one machine to the next one. The other is to store objects in order to be able to compensate for a downtime of the upstream machine and to prevent the immediate propagation of a tailback in case of a downtime of the downstream machine. In addition, the machines located upstream and downstream w.r.t. the filling machine work with higher output rates than the filler. This enables full upstream buffers and receptive downstream buffers to compensate for short downtimes of single machines.

These design principles help achieving a continuous operation of the filling machine. However, in practice, they cannot guarantee avoidance of unwanted idle time of the filler, and (unplanned) downtime of the plant can lie in the range of 10-30 percent.

Machine failures of significant duration, gaps caused by a large number of objects being sorted out, stoppages caused by toppled or jammed objects, or just mistakes of the operators result in downtime of the filling machine and decrease the availability of the entire plant. Because of the interlaced flows of the various object types, time offsets, and the large scale of the plants, the reasons for such plant downtimes can be difficult to identify by the plant operators, particularly since their number has been progressively reduced over the past years. In consequence, bottle filling and packaging industries are highly interested in an automated diagnosis tool for their plants.

There are a number of requirements and challenges to automated diagnosis raised by this application task. A fundamental economical condition is the fact that many of the potential end users, e.g. breweries, are small or medium enterprises, which could not afford spending many resources on the establishment or adaptation of a tailored diagnosis system for their plant. Another practical requirement is to cheaply accommodate frequent changes in the structure of the line, due to rearrangement or addition of machines. Both issues suggest a model-based solution to diagnosis, which allows performing adaptation by simply (re-)specifying the plant structure.

Additional arguments for such a solution stem from the facts that usually a plant is a combination of machines from various manufacturers with different instrumentation and available data and that there may be temporarily missing data due to technical problems. This requires a flexible solution that derives the best diagnosis from whatever data is available (in contrast, for instance, to decision trees based on a fixed set of observables).

Heterogeneity and changes of the set of machines also establishes a requirement on the model: firstly, it has to be machine-centered and compositional; secondly, it has to be stated at a level of abstraction that covers types of
machines, independently of specificities and the manufacturer. Besides these fundamental characteristics, the model has to be capable of properly predicting the propagation of gaps in the stream of objects (potentially causing a lack in supply to subsequent machines) and tailbacks caused by blockages, as well the propagation of special features and deficiencies of the transported objects, which may be caused by improper performance of one machine (e.g. improper cleaning) and may affect the (mis-)behavior of another element downstream (e.g. an inspection machine). The available data is inherently incomplete and imprecise. Even balance equations do not necessarily hold, because bottles may have been removed by an operator (for inspection or because they blocked the flow) or simply have fallen off the belt.

3. Models of Transportation Elements

3.1 Previous Work

The only similar work we are aware of (except for discrete-event-simulation models used for validation of the control, which do not lend themselves easily to model-based diagnosis) is in the domain of transport of paper in a copier. [Gupta-Struss 95] presents a process-oriented model, and [Fromherz et al. 03] develop a component-oriented model for control generation. Both models are compositional, but focus on the motion of individual sheets, rather than the more abstract perspective of flow of objects.

3.2 Modeling Assumptions

We first list the most important assumptions underlying the transportation models presented here, which are fulfilled in our project domain (under normal conditions), but should also apply to a much broader class of problems.

• The transported objects are rigid bodies with fixed spatial extensions and are not significantly deformed through transportation.
• They are transported with a fixed orientation (like crates), or the orientation does not affect transportation times significantly (e.g. due to a symmetric cross-section, as for bottles).
• There is no interaction among the objects or between objects and the components that has a significant impact on the transportation process (such as bouncing).
• Objects can move only in the direction of the motion of the transportation means (or not at all), although not necessarily with the same speed.

3.3 A Model of a Transportation Element with Buffer

In order to present the essentials of the modeling approach, we consider some sort of archetype of model, which can be specialized or extended to accommodate other kinds of machines. This is a machine that
• has one input and one output with \( v_{in}, v_{out} \) being the respective speeds of the means for transportation (e.g. belts),
• possibly transforms or modifies one kind of object (as, for instance, cleaning of bottles), but does not amalgamate several objects to form a new one,
• has a buffer with a (constant) capacity \( C \).

The process of buffering the objects can be fairly random, as illustrated by the bottle conveyor in Figure 3, where bottles may gather in bulk. However, it is assumed, that (under normal behavior) no object is prevented from approaching the output unless it is blocked by other objects ahead, waiting for output. For instance, within the bottle conveyor, its shape and several parallel belts with different speeds ensure that bottles are not left in some corner, but pushed towards the “ideal” fastest belt, if there is space. The intuition behind the model can be best described in terms of three fundamental concepts and five “behavior rules”, each of which is first introduced informally and then turned into equations. As stated before, one of the problems to be solved stems from the fact that a local machine model in isolation cannot determine whether an actual flow occurs at its input and output. But it can and has to express the limits on the machine’s potential to take in or output objects. This is reflected by

**Concept 1** The potential input and output flow, \( \text{in}, \text{out} \), represent the maximal flows the machine can accept or generate, dependent on its internal state. The actual flows are represented by two different variables, \( \text{in}, \text{out} \). The first restriction is determined by

**Rule 1** The potential input flow is given by the input speed of the transportation element, unless the buffer is full. In this case, it cannot be higher than the actual output flow.

In the mathematical model (see Fig. 4), this rule is formalized by equation 1, where \( d \) denotes the diameter of the object cross-section and \( B \) is the filling degree of the buffer (in terms of number of objects). It involves the assumption that an actual outflow generates the potential for intake instantaneously, which is not true in practice and, hence, another reason for expressing tolerance intervals with values and time. Note that we take all speeds and flows as positive, as their sign is determined by their association with the intrinsic direction of the transportation element. Computing \( B \) is straightforward.

**Rule 2** The change in the total number of buffered objects is determined by the actual input and output flows.

The respective equation 2 indicates that \( B \) will be computed by integrating the difference of the actual flows. Setting up the model fragments for the potential output flow is based on the second key idea:
Transportation Element with Buffer

**State variables**
- \( B(t) \): # objects in buffer
- \( B_{out}(t) \): # objects buffered for immediate output
- \( v_{in}(t) \): velocity of input transportation means
- \( v_{out}(t) \): velocity of output transportation means
- \( t_q(t) \): minimal transportation time

**Parameters**
- \( d \): diameter of transported object (in transportation plain)
- \( C \): Capacity (as number of objects)

**Interface variables**
- \( \text{in}.q_{in}(t) \): potential inflow [objects/s]
- \( \text{out}.q_{out}(t) \): potential outflow [objects/s]
- \( \text{in}.q_{in}(t) \): actual inflow [objects/s]
- \( \text{out}.q_{out}(t) \): actual outflow [objects/s]

**Equations**

1. \( \text{in}.q_{in}(t) = \frac{v_{in}(t)}{d} \) if \( B(t) < C \)
2. \( \text{in}.q_{in}(t) = \min (v_{in}(t)/d, \text{out}.q_{out}(t)) \) if \( B(t) = C \)
3. \( \text{out}.q_{out}(t) = \min (\text{in}.q_{in}(t) - t_q, v_{out}(t)/d) \)
4. \( \frac{d}{d\tau} \text{out}.q_{out}(t) = \text{in}.q_{in}(t) - t_q \) if \( B_{out}(t) \geq 1 \)

**Connector between Transportation Elements**

**Interface variables**
- \( T_{E_{in+1}}, \text{in}.q_{in}(t) \): potential inflow of upstream element \( T_{E_{in+1}} \)
- \( T_{E_{out}}, \text{out}.q_{out}(t) \): potential outflow of downstream element \( T_{E_{out}} \)
- \( T_{E_{in+1}}, \text{in}.q_{in}(t) \): actual inflow of upstream element \( T_{E_{in+1}} \)
- \( T_{E_{out}}, \text{out}.q_{out}(t) \): actual outflow of downstream element \( T_{E_{out}} \)

**Equations**

5. \( T_{E_{out}}, \text{in}.q_{in}(t) = \min (T_{E_{out}} \cdot \text{in}.q_{in}(t), T_{E_{out}} \cdot \text{out}.q_{out}(t)) \)
6. \( T_{E_{out}}, \text{out}.q_{out}(t) = T_{E_{max}}, \text{in}.q_{in}(t) \)

**Figure 4. Equations of buffer and connector**

Concept 2 \( B_{out} \) denotes the number of buffered output objects at time \( t \), i.e. the number of objects that can possibly be subject to output at this time.

Before we clarify this crucial concept, we use its intuitive understanding and the third concept for formulating the rule for the potential output flow.

Concept 3 The minimal transportation time, \( t_q \), is the time an object needs to get directly from the input to the output, i.e. if it is not delayed by other objects that are piling up.

In case of the bottle conveyor, this means that the bottle stays on the fastest (innermost) belt.

Rule 3 The potential output flow is determined solely by the output speed, if there is more than one buffered output object. Otherwise, it cannot be higher than the actual input flow at the time reduced by the minimal transportation time.

One should be aware that in the second case, each single object may (potentially) leave the output with the speed \( v_{out} \). However, if the input flow at the time when it entered was lower, there will be a gap occurring after the output of the object, which makes the (average) flow lower than \( v_{out} \).

As a special case, the potential output flow becomes zero, if the actual input flow was zero at the respective time. Again, the respective equation 3 in Figure 4 formalizes this. Computing \( B_{out} \) also involves the minimal transportation time \( t_q \). If an object entered the transportation element later than time \( t - t_q \), it cannot possibly reach the output at time \( t \) and, hence, cannot become part of the buffered output objects. If it entered earlier, it may or may not have already left the output before \( t \), depended on how the actual output flow reduced \( B_{out} \). This consideration is captured by

**Rule 4** The change in the number of buffered output objects at time \( t \) is determined by the actual input flow at time \( t - t_q \) diminished by the actual outflow at time \( t \).

Hence, also \( B_{out} \) is obtained by integration according to equation 4, which completes the model of the transportation element with buffer. Note, that \( B_{out} \) is not necessarily the number of objects that form a contiguous pile in front of the output. It could be less, because the last objects that joined the pile entered later than \( t - t_q \).

3.4 Interaction of Transportation Elements

What remains to be done is determining the actual flows from the potential flows of connected machines. This interaction is captured by a model of a generic connector used for connecting all types of transportation elements. The respective rule and equation 5 (Fig. 4) are straightforward:

**Rule 5** The actual output flow of a machine is limited by both its own potential output flow and the potential input flow of the following machine (and equal to the actual input flow of this machine).

3.5 Other Features and Transportation Elements

The buffer model leaves options for different use and specialization. Due to lack of space, we can only sketch some important cases, many of which are fairly straightforward. For instance, \( v_{in} \) and \( v_{out} \) could be different as for the entire bottle conveyor shown in Figure 3. In this case, the minimal transportation time \( t_q \) needs to be calculated or estimated based on varying speeds along the “ideal path”. Alternatively, the same conveyor can be considered as an aggregation of several buffers in series each with one unique speed on its fastest belt, which eases the computation of \( t_q \). Note that the speeds are subject to control and may vary dynamically. Therefore, in case of a unique speed, \( t_q \) is determined by the equation

\[
l = \int_{t - t_q}^t v(\tau) d\tau.
\]
where \( l \) is the length of the “ideal path” and \( v(t) \) its time-varying speed.

Gates may sit at the input or output of transportation elements and are controlled in a binary manner in order to block the flow entirely if necessary. This is captured by multiplying the respective speed with a factor of \( (1 - \text{state}_{\text{gate}}) \), if \( \text{state}_{\text{gate}} \) is 1 for a closed gate and 0 otherwise.

While the bottle conveyor has no fixed relation between the speed of the belts and the motion of the bottles, which may slide, other machines, such as the filler, transport objects by locking them to certain sockets. This is obtained as a specialization of the buffer model with a unique speed and the capacity given by the number of sockets that can be occupied by objects while processing them.

Some elements, such as the bottle cleaning unit, may have \( n \) inputs of the same type of objects). To accommodate this feature in the model, we simply have to replace the actual input flow by the sum of several individual input flows. Elements having several outputs (for objects of the same type) usually require some modeling of the mechanism that distributes the objects among the various outputs, e.g. evenly (if possible) or according to some criteria. An example for the latter case is given by inspection machines ejecting objects that fail to pass some test.

Another class of machines produces an output by combining objects of different kinds, as for instance the packaging of 20 bottles in a crate. The ratio of the number of different objects participating in this combination is usually not arbitrary, but exactly specified. This ratio links the various potential and actual inflows and the outflow, which is then limited by the “slowest” input flow (relative to the ratio of the respective object type).

The counterpart to this very generic combination element is the separation element, with unpackers being a subclass, in which the slowest actual outflow of a decomposition result limits the potential inflow of the composite object.

This set of fairly generic model types turns out to cover the variety of machines in a bottling plant and, more generally, also in the food packaging plants that we encountered.

4. Validation of the Base Model

In order to validate the component models described above, we implemented them as numerical simulation models in MATLAB/SIMULINK® [MathWorks 08] and compared the simulated behavior (using the solver ‘ode4’ (Runge-Kutta) with a fixed-step size of one second) with the one of real plants.

Every component was modeled using the equations introduced above and tested in isolation to check whether it was adequate of and stated in a context-independent manner, which is a prerequisite for compositionality. In a second step, a model of a complete plant was configured using the validated components.

In testing the individual components, values of single parameters and variables were varied, and the response of the simulated behavior was monitored. For example, the predicted changes in the buffered material B of a component for different values of the input speed \( v_{in} \) and the output speed \( v_{out} \) are shown in Figure 5. It depicts that the buffer fills as long as the input speed is higher than the output speed (assuming a sufficient supply), whereas with the input speed reduced to its minimum 0.1 and the output speed being still high, the amount of buffered objects decreases.

Because of the minimal transportation time, \( t_{b} \) of the component, the buffer is not completely emptied, as long as there is input available. Furthermore, only the objects represented by the variable \( B_{out} \) determine the existence of an output flow. Another real characteristic behavior can be reproduced when increasing the input speed while maintaining the output speed constant. Although \( v_{in} \) is still higher than \( v_{out} \), the buffer filling degree remains constant after a certain time, because it is limited by the maximum capacity of the component.

Similar results were achieved by testing the other component type models, providing evidence that the models capture the features relevant to the diagnostic task and do not violate context-independence.
The second challenge was validation by comparing the simulated behavior of a plant model with the behavior of a real plant. Several test cases were constructed, based on real-world downtimes scenarios of the bottling plant whose topology is shown in Fig. 6. The simulated plant consists of a primary flow of bottles and a secondary object flow of crates. In one test case, the downtime propagation of a failure of the crate washer was simulated and analyzed. This failure interrupts both object flows. After some delay, missing input occurs at the crate packer. Also the unpacker stops at some point, due to its output being blocked. The details of the propagation of failure depend on the capacities and filling degrees of the various buffers connecting the machines. For instance, if the crate magazine is empty and all other buffers are filled with a sufficient degree, the lack of crates will rapidly reach the crate packer. This causes a blockage of the labeling machine and the bottle filler (because the packer is not able to process the bottles) before the lack of bottles in the primary flow (caused by the inoperable unpacker) reaches the filling machine. In contrast, if the crate magazine is completely full, the crate packer keeps working for some time, and the filling machine will be stopped due to a lack of bottles.

Even for this complex scenario, the simulation model reproduces the behavior of the real world plant. Similarly, the characteristics of fault propagation occurring in real plants were predicted for other relevant scenarios.

5. Abstraction to Qualitative Diagnosis Models

Using the model presented above directly for diagnosis is not appropriate. Firstly, as for all numerical models, its accuracy is only a pretended one in many respects, e.g. in assuming conservation laws to hold and in ignoring the imprecision in the available data, e.g. when flows are determined via counters or the speed of belts. Secondly, the diagnostic task requires the analysis of qualitative, rather than arbitrarily small numerical deviations from the nominal behavior and, hence, needs to be addressed by an appropriate level of abstraction in the model.

The level of model abstraction depends on the intended goal of the diagnosis: we first focused on “hard” failures (stop of the filling machine, that is) caused by hard faults (blockage of another machine), which can be based on distinguishing zero from non-zero flow only. For capturing “soft” faults (deviating behaviors) that lead, perhaps in combination, to a hard failure or a non-optimal behavior, a different model is required.

5.1 Sign-based Absolute Model

The total interruption of the flow requires distinctions between zero and non-zero flows only. Sign abstraction of the numerical model yields the qualitative constraints on the variables shown in Fig. 7 (we omit equations (2) and (4), which are difficult or impossible to exploit because

<table>
<thead>
<tr>
<th>Transportation Element with Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) [in.q_out(t)] = [v_in(t)] if C-B(t) &gt; 0</td>
</tr>
<tr>
<td>[in.q_out(t)] = min ([v_in(t), [out.q_in(t)]]) if C-B(t) = 0</td>
</tr>
<tr>
<td>[in.q_out(t)] = [v_out(t)] [out.q_in(t)] [C-B(t)]</td>
</tr>
<tr>
<td>0 0 + +</td>
</tr>
<tr>
<td>0 + + +</td>
</tr>
<tr>
<td>0 0 + 0</td>
</tr>
<tr>
<td>0 + 0 0</td>
</tr>
<tr>
<td>(3) [out.q_out(t)] = [v_out(t)] if B_out(t)-1 ≥ 0</td>
</tr>
<tr>
<td>[out.q_out(t)] = min ([in.q_out(t-n-1), [v_out(t)])) if B_out(t)-1 &lt; 0</td>
</tr>
<tr>
<td>[out.q_out(t)] [v_out(t)] [in.q_out(t-n-1)] [B_out(t)-1]</td>
</tr>
<tr>
<td>0 0 + 0</td>
</tr>
<tr>
<td>0 + + 0</td>
</tr>
<tr>
<td>0 + + -</td>
</tr>
<tr>
<td>0 + 0 -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connector between Transportation Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5) [TE_out.q_out(t)] = min ([TE_in1-in.q_out(t), [TE_out.q_out(t)])</td>
</tr>
<tr>
<td>[TE_out.q_out(t)] = [TE_in1-in.q_out(t)]</td>
</tr>
<tr>
<td>[TE_out.q_out(t)] [TE_in1-in.q_out(t)] [TE_out.q_out(t)]</td>
</tr>
<tr>
<td>0 0 +</td>
</tr>
<tr>
<td>0 + +</td>
</tr>
</tbody>
</table>

Figure 7. Sign-based qualitative models of buffer and connector. |x| denotes the sign of x. “+” in a row represents “no restriction” and, hence, the entire row multiple tuples

neither B(t) nor B_out(t) can be observed properly together with the respective finite relations. (Remember that flows and speeds cannot be negative).

The abstraction of combination elements (such as the crate packer) outlined in section 3.5 will include the application of the three model fragments of Fig. 7 to all individual inflows as well as a constraint simply stating the qualitative equality of all inflows (the ratio of the flows drops out, because it is a positive number):

[in1,q_in(t)] = [in2,q_in(t)] = ... = [inN.q_in(t)].

This captures, for instance, the fact that one lacking input will stop all other inputs, as well. The dual applies to separation elements.

This model has been validated using the diagnosis tool RAZR’R [Raz’r 08] on several scenarios, including the one described at the end of section 4, which involves a fault in the washer. (Because the current version of RAZR does not support the required temporal indexing of the predictions, the temporal information was stripped off and cyclic prediction was prevented in order to avoid spurious inconsistencies due to different values occurring at
different times). The model is consistent with a lack of crates for the packer, which propagates backwards to a potential stop of the unpacker, which in turn may be caused by the inoperability of the washer.

We briefly demonstrate that the inferential power of the model suffices for handling the considered class of faults and failures despite its simplicity: assume that a transportation element $TE_n$ with a single speed, $v_n = v_{out}(t)$, produces an output, i.e. $[TE_n, out.q_{ref}(t)] = +$, but has no inflow, $[TE_n, in.q_{ref}(t)] = 0$. Then the constraints yield:

\[
[T_{E_n, out.q_{ref}(t)}] = + \quad (5) \quad [T_{E_n, out.q_{ref}(t)}] = +
\]

\[
[T_{E_n, out.q_{ref}(t)}] = + \quad (3) \quad [T_{E_n, v_{ref}(t)}] = [T_{E_n, v_{ref}(t)}] = +
\]

\[
[T_{E_n, out.q_{ref}(t)}] = + \quad \land \quad [T_{E_n, v_{ref}(t)}] = +
\]

\[
[T_{E_n, in.q_{ref}(t)}] = + \quad \land \quad [T_{E_n, in.q_{ref}(t)}] = 0
\]

\[
[T_{E_n, 1, out.q_{ref}(t)}] = 0 \quad \land \quad [T_{E_n, 1, v_{ref}(t)}] = +
\]

\[
[T_{E_n, 1, in.q_{ref}(t)}] = [T_{E_n, 1, in.q_{ref}(t)} - t_0] = 0.
\]

If $TE_{n-1}$ is operational, which implies $[TE_{n-1}, v_{ref}(t)] = +$, then

\[
[T_{E_{n-1}, out.q_{ref}(t)}] = 0 \quad \land \quad [T_{E_{n-1}, v_{ref}(t)}] = +
\]

\[
[T_{E_{n-1}, in.q_{ref}(t)}] = [T_{E_{n-1}, in.q_{ref}(t)} - t_0] = 0.
\]

This means, even without information about the buffers, the lack is propagated backwards across the models of correct elements (but will be consistent with a “block” mode, for instance) as expected.

### 5.3 Qualitative Deviation Model

The base model can also be used as the starting point for an abstraction that allows analyzing more subtle problems: the filling machines may not always be forced to stop operation, but, perhaps, run at reduced speed due to insufficient supply. For this purpose, the base model can be transformed into one that captures the propagation of deviations from some reference along the lines of [Struss 04]). A deviation of a variable $x$ is defined as

\[
\Delta x = [x_{act} - x_{ref}],
\]

i.e. the difference between the actual and some reference value, which may remain unspecified. Usually, the latter represents some optimal or nominal value. This definition plus the sign-based abstraction for deviation variables and dropping $B(t)$ and $B_{out}(t)$ transforms the base model into the deviation model of Fig. 8. Both the domain abstraction to signs and the projection that eliminates the buffer variables establish a true abstraction of the original model. Besides the analysis of reasons for suboptimal performance, such a model may be useful or even necessary for the diagnosis of filler stoppages. The reason is that the filler may be stopped not because its inflow is zero for a long time interval, but because the available inflow is less than the flow requested by its speed, i.e. $v_{in}(t)/ds$, and, hence, there is a gap in the supply and the filler is not supplied with a bottle for each sock, as required.

This model has not yet been validated in the diagnostic setting. However, we provide again some evidence for its inferential power. The “soft version” of the previous example states that the output and the speed of $TE_n$ do not deviate, but its inflow is too low. We obtain

\[
\Delta T_{E_{n}, out.q_{ref}(t)} = 0 \quad \land \quad \Delta T_{E_{n}, v_{ref}(t)} = 0
\]

\[
(1) \quad [\Delta T_{E_{n}, v_{ref}(t)}] = 0
\]

\[
\Delta T_{E_{n}, in.q_{ref}(t)} = 0 \quad \land \quad \Delta T_{E_{n-1}, out.q_{ref}(t)} = -
\]

\[
(5) \quad [\Delta T_{E_{n-1}, out.q_{ref}(t)}] = -
\]

\[
\Delta T_{E_{n-1}, out.q_{ref}(t)} = - \quad \land \quad \Delta T_{E_{n-1}, v_{ref}(t)} = 0
\]

\[
(3) \quad [\Delta T_{E_{n-1}, in.q_{ref}(t)} = t_0] = -.
\]

i.e. again, the deviation is propagated upstream.

---

6. Summary and Outlook

The validation has provided evidence that the models really capture the essential features of plant behavior we are interested in from a diagnostic perspective. However, we do not only have to cope with inaccurate values of quantities, such as flows, speeds etc. due to the actual process and the available measurements. Also the temporal inferences are not crisp. For instance, from zero output flow of a normally behaving machine during some time interval $i_1$, an earlier time interval $i_0$ can be inferred, in which zero input flow must have occurred. This means, in contrast to other temporal propagation schemes, the prediction cannot state that the flow was zero during the entire interval $i_0$, but only that there exists a subinterval $i' \subseteq i_0$ with zero flow, which has to be taken into account in the consistency check. Furthermore, propagation will lead to progressively larger time intervals, which prompts for an approach that uses observations interleaved with prediction to narrow down the intervals.

There are also different types of diagnostic tasks, such as our current focus, off-line post-mortem diagnosis (through analysis of stored data), on-line post-mortem diagnosis, and predictive diagnosis.

Finally, the project aims at a contribution to improving the general conditions through standardization of the data acquisition. Partners of the consortium are the originators of an existing standard that has now been widely accepted for bottling plants. This has now been extended on the one hand regarding data relevant to diagnosis and on the other hand generalizing it for food packaging plants. This will significantly improve the conditions for effective and easily adaptable diagnostic solutions.

References


[Raz’r 08] http://www.occm.de/

A Mixed Causality Approach to Residual Generation
Utilizing Equation System Solvers and Differential-Algebraic Equation Theory

Carl Svärd and Mattias Nyberg
Department of Electrical Engineering
Linköping University, SE-58183 Linköping, Sweden
{carl, matny}@isy.liu.se

Abstract

The FDI approach to model-based diagnosis is considered. We present a method for residual generation that combines integral and derivative causality, and also utilizes equation system solvers and theory of differential-algebraic equation systems. To achieve this, a framework for computation of variables from sets of dependent differential and/or algebraic equations is introduced. The proposed method is applied to a model of the gas flow in an automotive diesel engine. The application clearly shows the benefit of using a mixed causality approach for residual generation compared with solely integral or derivative causality.

1 Introduction

With the rising demand for reliability and safety of technical systems, fault diagnosis has become increasingly important. In the FDI approach to model-based fault diagnosis, a mathematical model of the system, together with measurements, is utilized to generate residuals, used to detect and isolate faults present in the system. One residual generation approach [Staroswiecki and Declerck, 1989] is to, by means of structural analysis, use a part of the model, i.e., a subset of equations, to compute a subset of the unknown variables from sets of dependent differential and/or algebraic equations. This fact gives rise to differential and algebraic equations which utilizes theory for solving and analyzing general differential-algebraic equations. In the proposed method, the causality of differential equations is defused and the way a differential equation is handled depends on the context in which the variables appear, the available tools for equation solving, the available tools for approximate differentiation of measurements, and knowledge about initial conditions.

The paper is organized as follows. Section 2 presents preliminaries and some basic theory and references for differential-algebraic equations and structural analysis. In Section 3, a framework for computation of variables from sets of dependent differential and algebraic equations is presented. Sections 4 to 6 presents the proposed method. In Section 7, an application example clearly shows the benefits of using a mixed causality approach compared with either integral or derivative causality. Section 8 concludes the paper. Due to the limitation of space, proofs are omitted but can be found in [Svärd and Nyberg, 2008].

2 Preliminaries

Consider a model $M(E, X, Z)$ or $M$ for short, consisting of a set of equations $E = \{e_1, \ldots, e_m\}$ relating a set of unknown variables $X = \{x_1, \ldots, x_n\}$, and a set of known variables $Z = \{z_1, \ldots, z_p\}$. Introduce a third set, $D = \{\tilde{x}_1, \ldots, \tilde{x}_m\}$, containing the derivatives of the variables in $X$. Without loss of generality, we assume that the equations in the set $E$ are in the form

$$e_i : \quad f_i (\dot{x}, x, z) = 0, \quad 1 \leq i \leq m$$

(1)

where $\dot{x}$, $x$, and $z$ are vectors of the elements in the sets $D$, $X$, and $Z$, respectively.

Define the set of trajectories of variables in $Z$ that are consistent with the model $M(E, X, Z)$ as

$$O(M) = \{z : \exists x; f_i (\dot{x}, x, z) = 0, 1 \leq i \leq m\}.$$  

(2)

The set $O(M)$ is referred to as the observation set of the model $M$. A residual generator is here formally defined as follows.

Definition 1 (Residual Generator for $M(E, X, Z)$). A system with input $z$ and output $r$ is a residual generator for the model $M(E, X, Z)$ and $r$ is a residual if $z \in O(M) \Rightarrow r = 0$.

2.1 Differential-Algebraic Equation Systems

It is assumed that the model (1) contains both differential and algebraic equations, that is, it is a differential-algebraic equation (DAE) system, or descriptor system. DAE-systems appear in large classes of technical systems like mechanical-, electrical-, and chemical systems. Further, DAE-systems are also the result when using physically based object-oriented modeling tools, e.g., Modelica, [Mattson et al., 1998].

---

*This work was sponsored by Scania CV AB and VINNOVA (Swedish Governmental Agency for Innovation Systems).*
A common approach when analyzing and solving general DAE-systems, is to first seek a reformulation of the original DAE into a simpler and well-structured description with the same set of solutions, see [Kunkel and Mehrmann, 2006], and [Brenan et al., 1989]. To classify how difficult such a reformulation is, the concept of index has been introduced. These different index concepts depending on what kind of reformulation that is sought. In this paper we will use the differential index, which is defined as the number of times that all or parts of the DAE must be differentiated with respect to time in order to write the DAE as an ordinary differential equation (ODE), see for example [Brenan et al., 1989]. The reformulation thus aims to write the original DAE as an ODE, i.e. a system in state-space form.

2.2 Structure of the Model

Let $C \subseteq E$ and introduce the notations:

$$\varphi(C) = \{ x_j \in X : \exists x_i \in C, \frac{\partial f_i}{\partial x_j} \neq 0 \lor \frac{\partial f_j}{\partial x_j} \neq 0 \} ,$$

$$\varphi_d(C) = \{ x_j \in D : \exists x_i \in C, \frac{\partial f_i}{\partial x_j} \neq 0 \} .$$

Let $G = (E, X, A)$ be a bi-partite graph where $E$ and $X$ are the (disjoint) sets of vertices, and

$A = \{ (e_i, x_j) : x_j \in \varphi(x_i) \land e_i \in E, x_j \in X \} . \tag{3}$

The set of arcs. We will call the bi-partite graph $G = (E, X, A)$ the structure of the model $M(E, X, Z)$. Note that with this representation, there is no structural difference between the variable $x_j$ and the differentiated variable $\dot{x}_j$. An equivalent representation of $G$ is the bi-adjacency matrix defined as

$$B = \{ b_{ij} \colon b_{ij} = 1 \text{ if } (e_i, x_j) \in A, 0 \text{ otherwise} \} . \tag{4}$$

A matching $\Gamma$ on the bi-partite graph $G$ is a subset of $A$ such that no two arcs have common vertices. A matching with maximum cardinality is a maximum matching. A matching is a complete matching with respect to $E$ (or $X$), if the matching covers every vertex in $E$ (or $X$).

By directing the arcs contained in a matching on the bi-partite graph $G$ in one direction, and the remaining arcs in the opposite direction, a directed graph can be obtained from $G$. A directed graph is said to be strongly connected if for every pair of vertices $x_i$ and $x_j$ there is a directed path from $x_i$ to $x_j$. The maximal strongly connected subgraphs of a directed graph is called its strongly connected components (SCC), see for example [Asratian et al., 1998].

There exists a unique structural decomposition of the bi-partite graph $G = (E, X, A)$, referred to as the Dulmage-Mendelsohn (DM) decomposition, [Dulmage and Mendelsohn, 1958], [Murota, 1987]. It decomposes $G$ into irreducible bi-partite subgraphs $G^+ = (E^+, X^+, A^+)$, $G_0^0 = (E_0^0, X_0^0, A_0^0)$, $1 \leq i \leq s$, and $G^- = (E^-, X^-, A^-)$, referred to as DM-components, see Figure 1. The component $G^+$ is the over-determined part of $G$, $G_0^0 = \bigcup_{i=1}^s G^0_i$ the just-determined part, and $G^-$ the under-determined part. The DM-components $G_0^0 = (E_0^0, X_0^0, A_0^0)$ correspond to the SCC of the directed graph induced by any complete matching on the bi-partite graph $G_0^0$, [Murota, 1987].

3 Computability of Variables

Introduce the notation $X_I$ for the subset of $X$ defined as $X_I = \{ x_i : i \in I \}$, where $I \subseteq \{ 1, \ldots, n \}$. A similar convention will be used to denote subsets of $D, Z, E$. Also, $I$ will be used to denote the complement of the set $I$ in $\{ 1, \ldots, n \}$, i.e. $\bar{I} = \{ 1, \ldots, n \} \setminus I$. To retrieve the indices of a set of variables (or equations), the operator $\operatorname{ind}(\cdot)$ is introduced, i.e. $\operatorname{ind}(X_I) = I$. Now, let $I \subseteq \{ 1, 2, \ldots, n \}$ and $J \subseteq \{ 1, 2, \ldots, m \}$, and consider the sets $X_I$ and $E_I$.

![Figure 1: The bi-adjacency matrix showing the DM-decomposition of $G$. The line along the diagonal corresponds to a maximum matching on $G$.](image)

**Definition 2** (Computability). The variables $x_I$ are computable from the equations $E_I$ if, given trajectories of the variables $X_I \cup Z$, trajectories of $X_I$ can be computed with the available tools.

3.1 Tools for Computation of Variables

Computability of a set of variables from a set of equations generally depends not only on the analytical properties of the equations in the set, but also on the set of tools that are available for use. In this paper, three types of tools are considered:

**DE Solving Tools**: Tools for solving explicit ordinary differential equations;

**AE Solving Tools**: Tools for solving algebraic (not differential) equation systems;

**Differentiating Tools**: Tools for approximate differentiation of measured (known) variables.

An AE solving tool is typically some software package for symbolic or numerical equation solving. A differentiating tool can for example be an implementation of a low-pass filter or a smoothing-spline approximate differentiator, [Wei and Li, 2006]. In this paper, we assume that AE solving tools are available through existing standard software packages like e.g. Maple or Mathematica, and design and implementation of AE solving tools will not be considered. We also assume that DE solving tools are always available, i.e. that the states of an explicit ordinary differential equation (a DAE of differential index 0) can be computed if the initial conditions of the states are known and consistent. This can be motivated by the fact that there exist several efficient methods for solving ODEs, see for example [Brenan et al., 1989]. Implementations are available in for example MATLAB and SIMULINK. Of course the assumption is not always valid and numerical solving of ODEs involves difficulties and problems such as stability and stiffness but this is not in the scope of this paper.

**Proposition 1** (Computability). The variables $X_I$ are computable from the equations $E_I$ if

1. the available AE solving tools admits a transformation of $E_I$ into

$$\dot{x}_I = \dot{g}_I(\dot{x}_I, x_I, x_{I^+}, x_{I^-}, z) \quad \tag{5a}$$

$$x_{I^+} = \dot{g}_I(\dot{x}_I, x_I, x_{I^-}, z) , \quad \tag{5b}$$

where $I^d = \operatorname{ind}(\varphi_d(E_I)) \cap I$, and $I^s = I \setminus I^d$.

2. the initial conditions of the variables in $X_{I^s}$ are known and consistent, and

3. the derivatives in $\varphi_d(E_I) \cap D_I$ can be obtained with the available differentiating tools.

**Remark 1**. If all equilibrium points of the system (5a) are, or with for example state-feedback can be made, (globally) asymptotically stable, the effect of the initial conditions are neglectable and condition 2 can be removed, see for example [Khalil, 2002].
Remark 2. One alternative to differentiate unknown variables directly, is to propagate known variables through a set of equations so that derivatives of unknown variables can be expressed as derivatives of known, i.e. measured, variables. Assume for example that we want to compute the derivative \( \dot{x}_1 \) and we also have that \( x_2 = z_1 \). To compute \( \dot{x}_1 \), we use a differentiating tool to compute \( \dot{z}_1 \) and then use \( \dot{x}_1 = \dot{z}_1 \).

There are two important special cases of computability. If the variables \( X_I \) are computable from \( E_J \) and \( I^a = I, I^d = \emptyset \), i.e.

\[
x_I = g(\hat{x}_I, x_I, z_I)
\]

(6)

the variables \( X_I \) are said to be algebraically computable from \( E_J \). Conversely, if \( I^d = I \) and \( I^a = \emptyset \), i.e.

\[
\hat{x}_I = g(\hat{x}_I, x_I, z_I)
\]

(7)

the variables \( X_I \) are said to be differentially computable from \( E_J \). If a set of variables is algebraically computable, so called derivative causality is used, and if a set of variables is differentially computable integral causality is used, see [Blanke et al., 2003]. Thus, if a set of variables is computed according to (5), or if a subset of variables in a model is algebraically computable and another subset of variables is differentially computable, both integral and derivative causality is used, i.e. mixed causality.

Remark 3. If the variables \( X_I \) are regarded as known variables and the sets \( I^d \) and \( I^a \) are both non-empty, (5) is equivalent to a semi-explicit DAE of differential index 1. Furthermore, (6) corresponds to an algebraic equation or equivalently an explicit DAE of differential index 1, and (7) to an explicit ODE or an explicit DAE of differential index 0, see [Brenan et al., 1989].

3.2 Initial Conditions and Estimation of Derivatives

The availability of initial conditions in general depends on the knowledge about the underlying system represented by the model. For complex physical systems, object-oriented modeling tools, e.g. Modelica [Mattson et al., 1998], are frequently used to build models. Often, this leads to that differentiated variables in the models correspond to physical quantities such as pressures and temperatures, which makes initial conditions known.

If the derivatives of a set of variables can be computed or not, depends both on the available set of differentiating tools and the quality of the measurements of the known variables. There are several approaches for approximate differentiating, e.g. smoothing spline approximation [Wei and Li, 2006]. An extensive survey of methods can be found in [Barford et al., 1999]. Derivative estimation is not in the scope of this paper, and will not be further considered.

4 A Method for Residual Generation

One approach to residual generation for a model is to sequentially compute subsets of the unknown variables from subsets of the equations, and then use an unused equation as residual. The generation of a residual will then consist of a finite sequence of variable computations, ending with an evaluation of a residual equation. The computation of variables in each step can thus only use variables that has been computed in some previous step, and known variables. To describe which variables that should be computed from which set of equations and in which order the variables should be computed, we introduce the concept variable set matching.

4.1 Variable Set Matching

Assume that \( I = \{I_1, \ldots, I_m\} \) and \( J = \{J_1, \ldots, J_n\} \) are partitions of \( \{1, \ldots, n\} \) and \( \{1, \ldots, m\} \) respectively, and let \( x \in \{X_{I_1}, \ldots, X_{I_m}\} \) be denoted \( X = \{X_{I_1}, \ldots, X_{I_m}\} \) and \( E = \{E_{J_1}, \ldots, E_{J_n}\} \).

Let \( \Lambda \) be a function from \( X \to E \) and assume that \((X_{I_1}, E_{J_1}) \in \Lambda \) and \((X_{I_j}, E_{J_k}) \in \Lambda \). Define the binary relation \( \prec \) on \( X \times E \) such that \((X_{I_j}, E_{J_k}) \prec (X_{I_1}, E_{J_1}) \) iff \( X_{I_1} \cap \text{var}(E_{J_1}) \neq \emptyset \)

Definition 3 (Variable Set Matching). The function \( \Lambda \) is a variable set matching for \( X \) on \( E \) if

1. \( \Lambda \) is injective,
2. for every \((X_{I_1}, E_{J_1}) \in \Lambda \) it holds that the variables \( X_{I_1} \) are computable from \( E_{J_1} \), and
3. the directed graph defined by \( \prec \) on \( \Lambda \) contains no directed cycles.

Remark 4. The first property ensures that \( \Lambda \) is complete with respect to the variable set \( X \). The third property prevents that computation of the variables in \( X_{I_1} \) requires the variables in \( X_{I_2} \) in which turn requires the variables in \( X_{I_3} \).

Proposition 2. The variables \( X \) are computable from the equations \( E \) if there exist partitions of \( X \) and \( E \) such that there exists a variable set matching \( \Lambda \) for \( X \) on \( E \).

The binary relation \( \prec \) on the variable set matching \( \Lambda \) defines a computation order for \( X \) on \( E \). A computation order can thus be represented as a directed acyclic graph.

4.2 Computation Sequence

If the variables \( X \) are computable from \( E \), the variable set matching \( \Lambda \) specifies which variables that should be computed from which equations. The order in which the variables in \( X \) must be computed is specified by the computation order \( \prec \). From a computation order, a computation sequence can be obtained.

Definition 4 (Computation Sequence for \( X \) on \( E \)). A linear order obtained by topological ordering of the directed (acyclic) graph defined by \( \prec \) on the variable set matching \( \Lambda \) is a computation sequence for \( X \) on \( E \).

In general, a computation sequence obtained from a computation order is not unique. Assume that the variables \( X \) are computable from \( E \), and that \( X = \{X_{I_1}, \ldots, X_{I_m}\} \) and \( E = \{E_{J_1}, \ldots, E_{J_n}\} \) are the partitions of \( X \) and \( E \) for which a variable set matching \( \Lambda \) exists. If we define \( R = \{1, \ldots, n\} \setminus \bigcup_{J \in \Lambda} J \), the set \( R \) will contain those equations that are not used in the computation of the variables in \( X \), and will be referred to as the redundant equation set associated with the variable set matching \( \Lambda \).

By using trajectories of the known variables in \( Z \) and the equations in \( E \setminus R \), trajectories of all variables in \( X \) can be computed according to the computation sequence. As the trajectories of all variables in \( X \) are computed, we can compute a residual from a redundant equation \( e_i \in E_R \) as \( r = f_i(x, x, z) \). The equation \( e_i \) will be referred to as the residual equation. We have motivated the following proposition.

Proposition 3. A computation sequence for \( X \) on \( E \) together with an equation \( e_i \in E_R \) is a residual generator for \( M(E, X, Z) \).

To illustrate the concepts presented above, we study a small academic example.

Example 1. Consider the following set of equations

\[
\begin{align*}
& e_1 : \dot{x}_1 + x_1 x_2 + x_3 + z_1 = 0 \\
& e_2 : \dot{x}_2 + x_1 + x_2 + x_3 + z_2 = 0 \\
& e_3 : \dot{x}_3 + x_3 - x_4 = 0 \\
& e_4 : x_3 + x_4 + x_5 + z_3 = 0 \\
& e_5 : x_5 + z_4 = 0 \\
& e_6 : h(x_1, x_4, z_5) = 0,
\end{align*}
\]

where it is assumed that neither \( x_1 \) nor \( x_4 \) can be computed from \( e_6 \). Let \( E = \{e_1, e_2, e_3, e_4, e_5, e_6\} \), \( X = \{x_1, x_2, x_3, x_4, x_5\} \), undefined, and \( I^d = I^a = \emptyset \).
\{x_1, x_2, z_1, z_3, z_4, z_5\}, \quad Z = \{z_1, z_2, z_3, z_4, z_5\}, \quad \text{and} \quad D = \{\dot{x}_1, \dot{x}_2, \dot{z}_1, \dot{z}_3\}.

By first studying the equations \(e_1\) and \(e_2\), we see that \(\{x_1, x_2\}\) can be (differentially) computed from \(\{e_1, e_2\}\), if the initial conditions of \(x_1\) and \(x_2\) are known and consistent, and the available AE solving tools admit that \(\dot{e}_1\) can be solved for \(\dot{x}_1\) and \(\dot{e}_2\) for \(\dot{x}_2\). We also see that if \(\dot{e}_3\) can, with the available AE solving tools, be solved for \(\dot{x}_3\) and \(\dot{e}_4\) for \(\dot{x}_4\), the equation set \(\{e_3, e_4\}\) becomes

\[
\begin{align*}
\dot{x}_3 &= -x_3 + x_4 \\
\dot{x}_4 &= -x_3 - x_5 - z_3,
\end{align*}
\]

which is on the form (5). Thus, if also the initial condition of \(x_3\) is known, \(\{x_3, x_4\}\) are computable from \(\{e_3, e_4\}\). If we assume that our AE solving tools admits that \(\dot{e}_5\) is solved for \(\dot{x}_5\), \(\{x_5\}\) is (algebraically) computable in \(\{e_5\}\). With \(X\) and \(E\) partitioned as \(X = \{x_1, x_2, x_3, x_4, x_5\}\) and \(E = \{\{e_1, e_2\}, \{e_3, e_4\}, \{e_5\}\}\), we now define the function \(\Lambda = \{(x_1, x_2), (x_3, x_4), (x_5, e_3), (x_5, e_4), (x_5, e_5)\}\) from \(X\) to \(E\). Since \(\{x_5\} \cap \text{varx} \{\{e_3, e_4\}\} = \{x_5\} \cap \{x_3, x_4, x_5\} = \{x_5\}\), it holds that \(\{x_5, \{e_3, e_4\}\} \prec (\{x_3, x_4, 1\}, \{e_3, e_4\})\), and by similar calculations, we conclude that \(\{x_1, x_2\} \prec (\{x_1, x_2\}, \{e_1, e_2\})\), and \(\{x_3, x_4\} \prec (\{x_3, x_4\}, \{e_1, e_2\})\). The directed graph defined by \(\prec\) on \(\Lambda\) is pictured below.

Since the directed graph contains no directed cycles, the function \(\Lambda\) is injective, and all variables are computable in respective equations for each element of \(\Lambda\), we conclude that \(\Lambda\) is a variable set matching for \(X\) on \(E\). From the directed graph, we obtain the computation sequence

\[
\{(x_3, x_4), \{e_3, e_4\}\} \rightarrow \{(x_1, x_2), \{e_1, e_2\}\}.
\]

The variables in \(X\) can then be computed in the order specified in (8). The only redundant equation in \(E\) is thus \(e_5\), and hence the residual is computed as \(r = h(x_1, x_4, z_5)\).

5 Finding Computation Sequences

The problem of designing a residual generator for the model \(M(E, X, Z)\) can be divided into the following steps:

1. Find a variable set matching;
2. Obtain a computation sequence from the computation order associated with the variable set matching;
3. Use a redundant equation as residual equation.

Step 2 is trivial, there are many efficient algorithms for topological ordering, see for example [Cormen et al., 2001]. Since also step 3 is trivial, the key point is to find a variable set matching.

5.1 Finding Variable Set Matchings

A variable set matching \(\Lambda\) for \(X\) on \(E\) is a function from a partition \(X = \{X_1, \ldots, X_s\}\) of \(X\) to a partition \(E = \{E_1, \ldots, E_j\}\) of \(E\), that fulfills the properties specified in Definition 3. To be more specific, it must hold that for every \(X_i \in X\) there exists \(E_j \in E\) such that \(X_i \subseteq E_j\) and that the directed graph defined by the relation \(<\) on \(\Lambda\) contains no cycles.

As said in Section 3, computability of variables from a set of equations depends on both the analytical properties of the equations in the set and the set of tools available for use. Naturally, a necessary condition for \(X_i\) to be computable from \(E_j\), is that \(X_i \subseteq \text{var}(E_j)\). Regarding the tools, we assume the following.

Assumption 1. AE solving tools require that \(|E_j| = |X_i|\).

Assumption 2. AE solving tools prefer, for e.g. numerical reasons, equation sets of small cardinality before equation sets with large cardinality.

An implication of Assumption 2 is that if the variables \(X_i\) are computable from \(E_j\), but there exists a variable set matching \(\Lambda = \{(X_{i_1}, E_{j_1}), \ldots, (X_{i_l}, E_{j_l})\}\) for \(X_i\) on \(E_j\), it is preferable to compute the variables \(X_i\) from the smaller equation sets \(E_{j_l}\).

Finding Equation Sets with Minimum Cardinality

Due to Assumption 2, we should find partitions of \(X\) and \(E\) with maximum cardinality. Thus, variable set matchings should contain equation (and variable) sets of minimum cardinality. However, equation sets of cardinality one can not always be used due to dependencies between equations. The dependencies will naturally induce cycles in the intended variable set matching.

Consider the bi-partite graph \(G = (E, X, A)\), representing the structure of the model \(M(E, X, Z)\) according to Section 2.2. Let \(I\) and \(J\) be subsets of \(\{1, \ldots, n\}\) respectively, such that the submodel \(M(E_I, X_I, Z)\) of \(M\) is just-determined. Let \(G = (E_I, X_I, A)\) denote the corresponding bi-partite graph representing the structure of \(M\). Motivated by the fact that the DM-components are irreducible bi-partite subgraphs, we apply the DM-decomposition to the graph \(G\) to obtain the DM-components \(G_i = (E_{I_i}, X_{I_i}, A_i)\). Since \(G\) is just-determined, the DM-components \(G_i\) are exactly the SCCs of the directed graph induced by any maximum matching on \(G\), see for example [Murota, 1987]. The following proposition holds.

Proposition 4. Let \(G = (E_I, X_I, A)\) be a just-determined part of \(G = (E, X, A)\) and \(G_i = (E_{I_i}, X_{I_i}, A_i)\), \(1 \leq i \leq s\) its strongly-connected components. The set

\[
\Lambda = \{(X_{i_1}, E_{j_1}), \ldots, (X_{i_l}, E_{j_l})\}
\]

is a variable set matching for \(X_i\) on \(E_j\) if for every \((X_{i_l}, E_{j_l}) \in \Lambda\), the variables \(X_{i_l}\) are computable from \(E_{j_l}\).

A justified question is then if there exists a variable set matching for \(X_i\) on \(E_j\), whose equation sets have less cardinality than the equation sets originating from the SCCs according to Proposition 4.

Proposition 5. Let \(G_i = (E_{I_i}, X_{I_i}, A_i)\) be a SCC of \(G\), then there exist no variable set matching with elements of cardinality larger than one for \(X_{I_i}\) on \(E_{j_l}\).

Proposition 5 implies that it is impossible to partition \(E_j\) into blocks with less cardinality than the SCC, without ending up with a cycle that prohibits a variable set matching.

Remark 5. SCCs are utilized in [Porté et al., 1988] and [Katsilis and Chantler, 1997] to determine the causal order [Iwasaki and Simon, 1986] of the variables in a model consisting of algebraic and differential equations. However, the causal order depends only on the occurrences of variables in the equations and does not consider computability, i.e. analytical properties of the involved equations, initial conditions, and available tools. SCCs are also used to partition sparse systems of equations into the so called BLT-form in tools for non-causal simulation, see for example [Frizion, 2004].

5.2 An Algorithm for Finding Variable Set Matchings

Proposition 4 states a sufficient condition for finding a variable set matching. Motivated by this and the implication of Proposition 5, we propose Algorithm 1 for finding a variable set matching for \(X_I\) on \(E_J\).

The function findAllSCC in Algorithm 1 returns equation and variable sets corresponding to the SCCs of the specified just-determined equation set, with respect to the specified set of variables. There are efficient algorithms for finding
Algorithm 1: findVariableSetMatching

Input: A just determined set of equations \( E_J \), a set of variables \( X_J \), a set of AE solving tools \( T_AE \), and a set of differentiating tools \( T_D \);

Output: A variable set matching \( \Lambda \) for \( X_J \) on \( E_J \);

\[ \Lambda := \text{findAllSCC}(E_J, X_J); \]

foreach \((E_J, X_J) \in S\) do

if isComputable\((E_J, X_J, T_AE, T_D)\) then

\( \Lambda := \Lambda \cup (X_J, E_J); \)

else

return \( \emptyset \);

end

end

Algorithm 1: findVariableSetMatching

Input: A just determined set of equations \( E_J \), a set of variables \( X_J \), a set of AE solving tools \( T_AE \), and a set of differentiating tools \( T_D \);

Output: A variable set matching \( \Lambda \) for \( X_J \) on \( E_J \);

\[ \Lambda := \text{findAllSCC}(E_J, X_J); \]

foreach \((E_J, X_J) \in S\) do

if isComputable\((E_J, X_J, T_AE, T_D)\) then

\( \Lambda := \Lambda \cup (X_J, E_J); \)

else

return \( \emptyset \);

end

end

5.3 Connection to MSO Sets

The problem of residual generation for a given model can, as said in beginning of this section, be divided into the three parts: 1) find a variable set matching, 2) obtain a computation sequence from the variable set matching, 3) use a redundant equation as residual equation. From the discussion above, it is clear that to find a variable set matching it is sufficient to consider a just-determined part of the given model. Hence, to design a residual generator it is sufficient to consider a part of the model that consist of a just-determined part and one redundant equation, that is, a minimal over-determined set of equations or in the structural case, a minimal structurally over-determined (MSO) set. The method for residual generation outlined in the beginning of this section, can thus be refined:

1. Find a MSO set in the model;
2. Find a variable set matching in the MSO set;
3. Obtain a computation sequence from the computation order associated with the variable set matching;
4. Use the redundant equation as residual equation.

There exist several efficient algorithms for finding all MSO sets in a model, see for example [Krysander et al., 2008].

6 Analyzing Computability of Variables

Consider the variable set \( X_J \) and equation set \( E_J \). From the development in Section 5.1, it is clear that we can limit our analysis to the case when \( |X_J| = |E_J| \), and \( E_J \) corresponds to a strongly-connected component.

The decomposition into strongly-connected components is based on the structural representation of the model adopted in Section 2. With this representation, there is no difference between a variable \( x_J \) and the corresponding differentiated variable \( \dot{x}_J \). The strongly-connected components therefore contains both differentiated and non-differentiated variables and thus both differential and algebraic equations. This means that the equation sets corresponding to SCCs are differential-algebraic equations. One approach for further analysis of computability of \( X_J \) from \( E_J \) is then to apply methods for analyzing and solving differential-algebraic equations (DAEs).

6.1 Analyzing Computability by Utilizing Differential-Algebraic Equation Theory

Motivated by theories for analyzing and solving differential-algebraic equation systems, we seek a reformulation, or transformation, of \( E_J \) into the form (5) according to Proposition 1. The ability to perform such a transformation, depends on the analytical properties of the equations in \( E_J \), as well as the available AE solving tools. Having obtained a transformation of \( E_J \) into one differential part (5a), and one algebraic part (5b), it is also desirable that both the differential and algebraic part are further decomposed into smaller just-determined parts, due to Assumption 1 and 2. We illustrate our approach for analyzing computability with an example.

Example 2. Consider the set of equations

\[
\begin{align*}
e_1 &: \dot{x}_1 - x_2 + x_1 x_4 + x_2 + x_3^2 + x_1 x_5 = 0 \\
e_2 &: \dot{x}_1 + x_2 + x_2 x_3 + 2x_3 x_5^2 = 0 \\
e_3 &: \dot{x}_3 - x_2^2 x_3 x_6 + x_7 = 0 \\
e_4 &: x_1 x_9 + x_4 + x_5 = 0 \\
e_5 &: x_2 + x_4 - x_5 + x_8 = 0 \\
e_6 &: x_2 x_3 x_5 + x_6 + x_9 = 0,
\end{align*}
\]

which for simplicity contains no known variables. The bi-adjacency matrix representing the structure of the equation set \( E = \{e_1, e_2, e_3, e_4, e_5, e_6\} \) with respect to \( X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\} \) is shown in (10). It is clear that \( E \) corresponds to a SCC of size 6.

<table>
<thead>
<tr>
<th>Equation</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( e_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( e_4 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( e_5 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( e_6 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Equation (10)

First consider the equation set \( \{e_1, e_2, e_3\} \), which contains the differentiated variables \( \{\dot{x}_1, \dot{x}_2, \dot{x}_3\} \). If we consider the structure of \( \{e_1, e_2, e_3\} \) with respect to \( \{x_1, x_2, x_3\} \), we obtain the bi-adjacency matrix shown in (11). We can now partition \( \{e_1, e_2, e_3\} \) into the equation sets \( \{e_1, e_2\} \), and \( \{e_3\} \), corresponding to SCCs of size two and one, with respect to the structure in (11).

<table>
<thead>
<tr>
<th>Equation</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_1 )</td>
<td>( \dot{x}_1 )</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>( \dot{x}_2 )</td>
</tr>
<tr>
<td>( e_3 )</td>
<td>( \dot{x}_3 )</td>
</tr>
</tbody>
</table>

Equation (11)

If our AE solving tools admits that \( \{e_1, e_2\} \) is transformed into

\[
\hat{x}_1 = \frac{1}{2}(-x_2 - x_2 x_3 - x_1 x_4 - x_3^2 - 2x_3 x_5^2 - x_7 x_9)
\]

(12a)

\[
\hat{x}_2 = \frac{1}{2}(x_2 - x_2 x_3 + x_1 x_4 + x_3^2 - 2x_3 x_5^2 + x_7 x_9),
\]

(12b)

and \( \{e_3\} \) into

\[
\hat{x}_3 = x_2^2 x_3 x_6 - \dot{x}_7,
\]

(13)

we have that \( \{x_1, x_2\} \) are differentially computable from \( \{e_1, e_2\} \), and \( \{x_3\} \) is computationally differentiable from \( \{e_3\} \) if the initial conditions of \( \{x_1, x_2\} \) and \( \{x_3\} \) are known and consistent and the derivative \( \{\dot{x}_7\} \) can be computed with the available differentiating tools.

Now instead turn to the equation set \( \{e_4, e_5, e_6\} \). From the bi-adjacency matrix in (10), we then see that \( \{e_4, e_5, e_6\} \) can be partitioned into the equation sets \( \{e_4, e_5\} \) and \( \{e_6\} \), which corresponds to SCCs of size two and one respectively. Under the assumption that our AE solving tools admits a transformation of \( \{e_4, e_5\} \) into

\[
x_4 = \frac{1}{2}(-x_2 - x_1 x_4 - \dot{x}_8),
\]

(14a)

\[
x_5 = \frac{1}{2}(x_2 - x_1 x_9 + \dot{x}_8),
\]

(14b)
and of \( \{ e_6 \} \) into
\[
x_6 = -x_2x_3x_5 - \dot{x}_9, \tag{15}
\]
we see that \( \{ x_4, x_4 \} \) are algebraically computable from \( \{ e_4, e_5 \} \) and \( x_6 \) is algebraically computable from \( \{ e_6 \} \), if the derivatives \( \{ \dot{x}_8 \} \) and \( \{ \dot{x}_9 \} \) can be computed with the available differentiating tools.

We have then transformed the original set of equations \( E \) into the form (5), with (12) and (13) corresponding to (5a), and (14) and (15) to (5b). Thus, we have \( I^d = \{ 1, 2, 3 \} \), \( I^a = \{ 4, 5, 6 \} \), and \( I = \{ 7, 8, 9 \} \). Hence, if the initial conditions of \( \{ x_1, x_2, x_3 \} \) are known and consistent, and the derivatives \( \{ \dot{x}_7, \dot{x}_8, \dot{x}_9 \} \) can be computed with the available differentiating tools, the variables \( X \) are computable from \( E \).

### 6.2 An Algorithm for Analyzing Computability

Motivated by Example 2 and the three conditions in Proposition 1, we propose the following procedure for analyzing if \( X_I \) are computable from \( E_J \).

1. Partition \( I \) into \( \{ I^a, I^d \} \), according to \( I^d = \text{ind}(\text{var}_D(E_J)) \cap I, I^a = I \setminus I^d \);
2. Determine if the initial conditions of the variables \( X_{I^a} \) are known and consistent;
3. Determine if the derivatives \( \text{var}_D(E_J) \cap D_I = \emptyset \);
4. Partition \( J \) into \( J^a \) and \( J^d \), such that \( \text{var}_D(E_{J^a}) \cap D_{I^d} = \emptyset \);
5. Find the SCCs of \( G^a = (E_{J^a}, X_{I^a}, A^a) \). For each SCC \( G^a_i = (E_{J^a_i}, X_{I^a_i}, A^a_i) \), determine if \( X_{I^a_i} \) can be computed from \( E_{J^a_i} \) with the available AE solving tools;
6. Find the SCCs of \( G^d = (E_{J^d}, D_{I^d}, A^d) \). For each SCC \( G^d_i = (E_{J^d_i}, D_{I^d_i}, A^d_i) \), determine if \( D_{I^d_i} \) can be computed from \( E_{J^d_i} \) with the available AE solving tools.

A complete, fully automated, algorithm can be found in Algorithm 2. The function \( \text{isInitCondKnown} \) determines if the initial conditions of the specified variables are available and consistent. The function \( \text{isAESolvable} \) determines if the specified variables are computable from the specified set of equations with the available set of AE solving tools. The function \( \text{isDifferentiable} \) determines if the derivatives of the specified variables can be computed with the available set of differentiating tools. The function regards propagation of derivatives as described in Remark 2.

Given ideal AE solving tools, ideal differentiating tools, and consistent initial conditions for the variables \( X_{I^d} \), where \( I^d = \text{ind}(\text{var}_D(E_J) \cap D_I) \), Algorithm 2 returns true iff the equation set \( E_J \) can be transformed into the form (5), possibly with either of the sets \( I^d \) or \( I^a \) empty. From this and Remark 3 it follows that with ideal AE solving tools, ideal differentiating tools, and consistent initial conditions for the variables \( X_{\text{ind}(\text{var}_D(E_J))} \), a variable set matching for \( X \) on \( E \) can be found with Algorithm 1 iff \( E \) is just-determined and its SCCs can be transformed to semi-explicit DAEs of differential index 1 or explicit DAEs of differential index 1 or 0, i.e. SCCs in the form (5), (6), or (7).

**Remark 6.** Although only SCCs corresponding to semi-explicit DAEs of differential index one can be handled with Algorithm 2, equation sets that are of higher differential index as a whole can be handled with the proposed method. Consider the equation set
\[
\begin{align*}
e_1 & : \quad \dot{x}_1 - x_2 = 0 \\
e_2 & : \quad \dot{x}_2 - x_3 = 0 \\
e_3 & : \quad x_1 - z_1 = 0,
\end{align*}
\]
which is a DAE of differential index 3, taken from [Mattson and Söderlind, 1993]. If we assume that our AE and differentiating tools are ideal, Algorithm 1 returns the variable set matching \( \Lambda = \{ (x_1, e_3), (x_3, e_2), (x_2, e_1) \} \), where each element corresponds to a SCC of size 1. The associated computation sequence \( (x_1, e_3), (x_2, e_1), (x_3, e_2) \) and the variables \( \{ x_1, x_2, x_3 \} \) can be computed from \( \{ e_1, e_2, e_3 \} \) as \( x_1 = x_2 = \dot{x}_1, \) and \( x_3 = \dot{x}_2 \). Thus, even though the original system is a DAE of differential index 3, the proposed method can be used to find a variable set matching since the SCCs of \( \{ e_1, e_2, e_3 \} \) are DAEs of differential index 1.

### 7 Application Example

In this section, the proposed method for residual generation is applied to a complex model of the gas flow in an automotive diesel engine.

#### 7.1 The Engine Model

The modeled engine is a six cylinder Scania diesel engine equipped with exhaust gas recirculation (EGR) and a variable geometry turbocharger (VGT). The model focuses on the gas flow in the engine and is described in [Wahlström, 2006]. To be better suited for residual generation, it was modified in [Kingstedt and Johansson, 2008]. The modified model contains in total 50 equations, 47 unknown variables, and 11 known variables. The variables represent physical quantities such as pressures, temperatures, and rotational speeds. The model consists of 8 differential equations and 42 algebraic equations, i.e. the model is a differential-algebraic equation.

#### 7.2 Configurations of the Algorithm

For comparison, the algorithm was applied to the engine model with three different configurations. The following parameters were used for configuration:

- Availability of initial conditions;
• Characteristics of AE solving tools;
• Characteristics of differentiating tools.

These parameters naturally influence the possibility to compute variables in different ways, and thus also the possibility to find variable set matchings.

The configurations used are shown in Table 1. With configuration $C_1$, the only way a set of variables can be computed from a set of differential equations is algebraically since no initial conditions are available, cf. (6). This is often referred to as integral causality, see [Blanke et al., 2003]. This approach for handling differential equations has been used in for example [Izadi-Zamanimadi, 2002] and [Dustegor et al., 2004]. With configuration $C_2$ on the other hand, the only way to compute a set of variables from a set of differential equations is according to (7), with the additional condition that $D_1 = \emptyset$, since no derivatives are available. This is in the literature referred to as integral causality, which is the way differential equations are handled in, for example [Palido and Alonso-Gonzalez, 2004] (still their framework supports the use of both integral and derivative causality). Configuration $C_3$ thus handles both integral and derivative causality and if a set of variables is computable from a set of equations depends on the analytical properties of the equations in the set and the available AE solving tools, according to Proposition 1. In all three configurations it is assumed that the AE solving tools only can handle equation sets with one element, i.e. SCCs of size one.

<table>
<thead>
<tr>
<th>Initial Conditions</th>
<th>AE Sol. Tools</th>
<th>Diff. Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>no</td>
<td>scalar equations yes</td>
</tr>
<tr>
<td>$C_2$</td>
<td>yes</td>
<td>scalar equations yes</td>
</tr>
<tr>
<td>$C_3$</td>
<td>yes</td>
<td>scalar equations yes</td>
</tr>
</tbody>
</table>

Table 1: Configurations of the algorithm

7.3 Results

By using an implementation of the engine model in MATLAB/SIMULINK, and a MATLAB implementation of Algorithm 1, the proposed method was applied to the engine model. The implementation utilizes the fact discussed in Section 5.3 and hence as a first step, all MSO sets are computed. This step was achieved with the toolbox described in [Frisk et al., 2006].

In total, 90 MSO sets were found in the engine model. In Table 2 it is shown in which of the MSO sets a variable set matching could be found with the different configurations of the algorithm. With configuration $C_1$, a variable set matching could only be found in one of the MSO sets, with configuration $C_2$ in four of the MSO sets, and with configuration $C_3$ a variable set matching could be found in 35 of the 90 MSO sets.

Detailed Study of a Specific MSO Set

We will now consider one of the MSO sets where a variable set matching could be found with configuration $C_3$, but not with the configurations $C_1$ and $C_2$. The MSO set, referred to as MSO set 4 in Table 2, contains 36 equations and 35 unknown variables. Of the 36 equations, only five are differential equations. By using an equation named $e_{36}$ as residual equation, a variable set matching could be found. The structure of the corresponding just-determined part of MSO 4 is shown in Figure 2.

The found variable set matching contains variable sets corresponding to 32 SCCs of size one, and one SCC of size three. The SCCs are marked with a square in Figure 2. The SCC of size three contains the variable set $\{T_1, T_e, x_r\}$ and equation set $\{e_{11}, e_{12}, e_{13}\}$, which are on the form $e_{11} : f_{11}(\dot{x}_r, x_r, T_1, W_{ei}, p_{im}, W_t, p_{em}) = 0$ $e_{12} : f_{12}(\dot{T}_1, T_1, T_e, x_r, T_{em}) = 0$ $e_{13} : f_{13}(T_1, T_e, x_r, W_{egr}, p_{im}, W_t, p_{em}) = 0$.

To compute $\{T_1, T_e, x_r\}$ from $\{e_{11}, e_{12}, e_{13}\}$, a scalar equation solver implemented in MATLAB was used to compute $\dot{x}_r$ from $e_{11}$, $\dot{T}_1$ from $e_{12}$, and $T_e$ from $e_{13}$. The equations $\{e_{11}, e_{12}, e_{13}\}$ could then be written on the form (5) and since the initial conditions of $x_r$ and $T_1$ were known, $\{T_1, T_e, x_r\}$ were computable from $\{e_{11}, e_{12}, e_{13}\}$.

In the SCC of size one corresponding to the equation set $\{e_1\}$, the variable $W_{egr}$ was algebraically computed using the differentiated variable $\dot{p}_{im}$. The derivative $\dot{p}_{im}$ was computed with a smoothing spline approximate differentiator implemented in MATLAB, and propagation of measured variables. In a similar way, the variables $W_t$ and $P_t$ were algebraically computed from $\{e_2\}$ and $\{e_21\}$, respectively. This was done by using the derivatives $\dot{p}_{im}$ and $W_t$.

Since integral causality were used to compute variables from the equation set $\{e_{11}, e_{12}, e_{13}\}$ and derivative causality to compute variables from the equation sets $\{e_{11}\}$, $\{e_{12}\}$, and $\{e_{21}\}$, it is clear that no residual generator could have been created from MSO set 4, with $e_{36}$ as residual equation, if either integral or derivative causality had been used.

8 Conclusions

We have presented a mixed causality approach to residual generation, that combines integral and derivative causality and also utilizes equation system solvers and theory of differential-algebraic equations. An important part of the proposed method is a framework for computation of variables from sets of dependent differential and/or algebraic equations. In the mixed causality approach, the way a differential equation is handled depends on the context in which variables appear, the available tools for equation solving and approximate differentiating of measurements, and knowledge about initial conditions.

Complete algorithms for finding residual generators with the proposed method, as well as analysis of computability of variables from dependent differential-algebraic equation systems, have been presented. The algorithms have been applied to a model of the gas flow in an automotive diesel engine. By applying three different configurations of the algorithm, corresponding to integral and derivative causality alone and mixed causality, it has been shown that considerably more residual generators can be found in the engine model with the mixed causality approach.
Table 2: A table showing in which of the MSO sets a variable set matching could be found with the different configurations of Algorithm 1.

References


Hypothesis Discrimination with Abstractions based on Observation and Action Costs

Gianluca Torta\textsuperscript{1}, Daniele Theseider Dupré\textsuperscript{2}, Luca Anselma\textsuperscript{1}

\textsuperscript{1} Università di Torino, Italy, email: \{torta,anselma\}@di.unito.it
\textsuperscript{2} Università del Piemonte Orientale, Italy, email: dtd@mfn.unipmn.it

Abstract

Several explanation and interpretation tasks, such as diagnosis, plan recognition and image interpretation, can be formalized as abductive reasoning and consistency reasoning. Some proposals address the problem based on a task-independent representation of a domain which includes an ontology or taxonomy of hypotheses. In this paper we rely on the same type of representation, and we address cost trade-offs in abduction intended as an iterative process where, like in model-based diagnosis, further observations are proposed to discriminate among candidates. Discrimination is performed up to an appropriate level which depends on the costs of actions (e.g. repair actions or therapy) to be taken based on the results of abduction, and on the cost of additional observations, which should be balanced with the benefits, in terms of more suitable actions, of better discrimination. Abstractions have a significant impact on this trade-off, given that the cost of observing the same phenomenon at different levels of abstraction may be quite different.

1 Introduction

Several explanation and interpretation tasks, such as diagnosis, plan recognition and image interpretation, can be formalized as abductive reasoning or related forms of nonmonotonic reasoning. A number of approaches [Chu and Reggia, 1991; Console and Theseider Dupré, 1994; Kautz, 1991], including recent ones [Besnard et al., 2007; Neumann and Möller, 2006], address the problem based on a representation of a domain which includes an ontology or taxonomy of hypotheses. Such a representation may have been developed independently of the reasoning task (in perspective, it may even be available on the Web in a shared ontology for different reasoning tasks), i.e., the structure reflects a natural representation of the domain, but it does not necessarily provide directly the best structure for diagnosis or interpretation.

In this paper we adopt a similar representation, and concentrate on the following issues:

- Dealing with abduction as an iterative process where, as in model-based diagnosis, further observations are proposed to discriminate among candidate explanations;
- Balancing the costs of observations with (reduced) costs or (increased) benefits of the results of abduction.

The costs/benefits associated with the results of abduction, in a diagnostic setting, correspond to the cost of repair actions or therapy, and are expected to decrease as long as more information is available on hypotheses; in a plan recognition or in an interpretation task, the human or software agent using the results should similarly achieve some benefit from a better discrimination of hypotheses or from more specific hypotheses, leading to a more focused action: this could either imply a reduced cost — e.g. if hypotheses are threats to the agent with costly defense actions — or an increased benefit — e.g. if the agent might use the results to earn money. In all settings, we intend that some action has to be taken based, in general, on the remaining candidate hypotheses. If the set of candidates is too broad or too abstract, the agent may incur into higher action costs due to (a combination of) the following reasons:

- more actions to be taken, to account for all possibilities, e.g. in component-oriented diagnosis, replacing all suspect components;
- selecting, for example, the action associated with the most probable explanation, with an expected cost which takes into account the cost of making, with a smaller probability, the wrong action (repairing the wrong part, taking the wrong therapy, defending from the wrong threat); and similarly, in the benefit case, making an action which will probably (but not certainly) be the right one for achieving the benefit.

The different issues are related: discrimination may be performed among hypotheses at the same level of abstraction, but it could also involve refining hypotheses. In any case, discrimination requires more observations, whose cost should be balanced with the benefits, in terms of more suitable actions, of better discrimination.

The presence of a domain representation with abstractions has a significant impact on this trade-off. The cost of observing the same phenomenon at different levels of abstraction may vary significantly; in fact, it may range from subjective information from a human (patient or user) to more or less costly medical or technical tests, or, in an image interpretation task, it may involve computationally complex image processing, to be performed interactively with the reasoning task, as...
Hypothesis Discrimination with Abstractions based on Observation and Action Costs

suggested in [Lamma et al., 1999].

In several settings, an observation which is itself expensive, because it consumes resources and time to be performed, may imply additional costs due to the delay before taking an action: breakdown costs in diagnosing a physical system, risk of death of the patient in medical diagnosis, taking defensive actions too late, missing the opportunity of earning money.

Moreover, if the knowledge base has been designed independently of the explanation/action task (e.g. diagnosis and repair), it could therefore include a detailed description of the domain which is not necessary for the task; more generally, the usefulness of a detailed discrimination may depend on the specific case at hand.

By explicitly considering abstractions in the iterative abduction process, we expect to reduce the observation costs significantly, yet maintaining the ability to exploit detailed observations and knowledge when convenient (similar advantages have been shown in inductive classification with abstractions, e.g. [Zhang et al., 2002]).

In the following, we first describe the knowledge we expect to be available. We then describe a basic iterative abduction loop and we concentrate on the criterion for selecting the next step in the loop: either performing a further observation at some level of detail, or stopping because the estimated most convenient choice is performing the action(s) associated with the current hypotheses. A later section is devoted to discussing the implications of allowing conjunctions of hypotheses (and observations) to be drawn from the same taxonomy. Finally, we summarize the contributions of the paper and conclude.

2 Domain Representation

The basic elements of the domain model are a set of abducibles (atomic assumptions) $A = \{A_1, \ldots, A_n\}$ and a set of manifestations $M = \{M_1, \ldots, M_m\}$. Each abducible $A_i$ is associated with an IS-A hierarchy $\Lambda(A_i)$ containing abstract values of $A_i$ as well as their refinements at multiple levels; similarly, each manifestation $M_j$ is associated with an IS-A hierarchy $\Lambda(M_j)$. We assume that the direct refinements of $A_i$ and $M_j$ are mutually exclusive, and at most one of the leaf values in a hierarchy is true in each situation, i.e. we allow at most one instance for each abducible and observation; moreover, for each leaf value $v$ of an abducible an a-priori probability $p(v)$ is given.

The hypothesis space $S(A)$ for the abduction task is the set of all of the combinations $\gamma$ of values drawn from one or more distinct hierarchies $\Lambda(A_i)$, while the manifestation space $S(M)$ is the set of all of the combinations $\omega$ of values drawn from distinct hierarchies $\Lambda(M_j)$. The relationships between the values of the abducibles and the values of the manifestations are defined by the domain knowledge $K \subseteq S(A) \times S(M)$.

Given an instance of manifestations $\omega \in S(M)$ and an instance of abducibles $\gamma \in S(A)$, $(\gamma, \omega) \in K$ means that $\omega$ is a possible observation set corresponding to hypothesis set $\gamma$.

We associate costs with the values of both abducibles and manifestations. Let $H \in \Lambda(A_i)$ be a value belonging to the IS-A hierarchy of $A_i$; its cost $ac(H)$ is the cost of the action that has to be taken when $A_i$ takes value $H$ (e.g. a repair action if $A_i$ represents a component and $H$ denotes one of its fault modes).

Let $h_1, \ldots, h_q$ be the children of $H$ in $\Lambda(A_i)$, i.e. the possible refinements of value $H$. We assume that:

$$\max\{\{ac(h_1), \ldots, ac(h_q)\}\} \leq ac(H) \leq \sum_{k=1}^{q} ac(h_k) \quad (1)$$

i.e. the action that we take for a value $H$ of $A_i$ costs no less than the most expensive action for its refinements and no more than taking the actions for all of such refinements. As for the manifestations, let $O \in \Lambda(M_j)$ be a value belonging to the IS-A hierarchy of $M_j$; its cost $ac(O)$ is the cost of making the observation which refines value $O$ into one of its children $\omega_1, \ldots, \omega_q$ in $\Lambda(M_j)$.

We can associate an action cost also with any instance $\gamma = \{H_1, \ldots, H_q\} \in S(A)$ of abducibles simply as $ac(\gamma) = \sum_{i=1}^{q} ac(H_i)$, i.e. we assume that independent actions are taken for each of the abducibles values that appear in $\gamma$.

With a slightly more complex computation we can also associate an action cost with a set of instances $\Gamma = \{\gamma_1, \ldots, \gamma_s\}$ representing the cumulative action cost if $\Gamma$ is the final set of explanations. In order to limit the cost analysis to the hypotheses in the current set of candidates, for each abducible $A_i$, we compute a new hierarchy $\Lambda(\{A_i\}, \Gamma)$ by considering the portion of $\Lambda(A_i)$ up to the least upper bound $LUB(\{A_i\}, \Gamma)$ that covers all of the values of $A_i$ that appear in $\Gamma$ and by further removing from such a sub-tree all of the values that do not appear in $\Gamma$.

In this way, it may happen that the cost $ac(H)$ of a value $H \in \Lambda(A_i, \Gamma)$ is greater than the sum of the costs $ac(h_k)$ of its children, since not all of the children of $H$ defined in $\Lambda(A_i)$ need to appear in $\Lambda(A_i, \Gamma)$. We therefore compute bottom-up modified costs $ac^*(\{A_i\}, \Gamma)$ which reestablish property (1). The action cost of $\Gamma$ is then computed just as:

$$ac(\Gamma) = \sum_{i=1}^{s} ac^*(LUB(\{A_i\}, \Gamma))$$

In Figure 1 we show a fragment of a fictitious medical domain model. On the left, there is the nosological description of some diseases, represented as three IS-A hierarchies of abducibles (with roots Disease1, Disease2, and Disease3). For example, Disease1.1 and Disease1.2 are two refinements of Disease1. On the right, there are possible symptoms and the possible medical examinations to be performed, represented as three IS-A hierarchies of manifestations (with roots Symptom1, LabTest1, and LabTest2).

The a-priori probabilities of the leaves of abducibles is assumed to be $\frac{1}{2}$, except $p(Disease1) = \frac{1}{3}$. The action costs of associated with each abducible are the costs of treating the disease; the observation costs of associated with each internal node of manifestation hierarchies are the costs of performing the related laboratory exam. The relationships between abducibles and manifestations are represented by rightwards dashed arrows. For example, Disease1 and all its more specific diseases implies LabTest2 to be positive; Disease1.2 implies LabTest1 to be positive, and
its refinements Disease1.2.1 and Disease1.2.2 imply more specific positive values of LabTest1. The relation K completes such explicit knowledge with negative values of tests as default values, e.g. the hypotheses set \{Disease3\} predicts \{Symptom1, LabTest1Neg, LabTest2Neg\}, and when refinements of hypotheses do not predict refinements of observations, they are intended as compatible with all refinements.

3 Iterative Abduction

We rely on the following loop for iterative explanation:

Input is a set of values \(\omega_I = \{O_1, \ldots, O_m\}\) representing the initial observations, i.e. the values of the set of manifestations \(M = \{M_1, \ldots, M_m\}\).

Generate a set \(\Gamma\) of candidates (i.e. explanations of \(\omega_I\)).

\[
\begin{align*}
\text{loop} & \quad O := \text{NextStep}(\Gamma); \\
& \quad \text{if } O = \text{STOP then exit} \\
& \quad \text{else} \\
& \quad \quad \quad \text{perform observation to refine } O \text{ into one of its children } o_k; \\
& \quad \quad \quad \Gamma := \text{Update}(\Gamma, o_k) \\
\end{align*}
\]

\(^1\)Without lack of generality, if no observation of \(M_i\) has been performed, \(\omega_I\) contains the root of \(\Lambda(M_i)\).

That is, we assume that one or more initial observations are given; that there is a way to generate candidate explanations based on them (see below), and to update candidates based on additional observations; and we proceed with selecting and performing one observation at a time, which, of course, is in general suboptimal, as the iterative process described in [de Kleer and Williams, 1987] for consistency-based diagnosis and modified in [Console et al., 1990] for abductive diagnosis.

The goal of this paper is to provide a general approach to the selection of the next step, and not to embrace a specific semantics of abduction and implementation of the candidate generation and update steps in the loop. Their concrete definition would depend on several issues. It could involve a mix of abductive and consistency reasoning depending on the completeness and predictiveness of knowledge [Console and Torasso, 1991; Theseider Dupr\’e, 2000]. Its formulation would also depend on the way \(K\) is represented — in particular, what is explicitly represented and what is implicitly intended; e.g. implicit default assumptions of normality like in the example in this paper would probably not be suitable for an image interpretation task.

Predictiveness of knowledge, and, in particular, whether a set of assumptions implies a single value for an observation or not (so that requiring abductive explanations which imply the observed value either makes sense or is too demanding)
is particularly relevant with hierarchies of assumptions and observations. In general, we should accept an explanation that implies some abstraction of the observation, rather than the observation itself [Kautz, 1991; Besnard et al., 2007], but we do not explicitly address this issue in this paper.

Another important issue when abstractions are involved is the fact that abstract as well as detailed assumptions may take part in explanations and there may be too many detailed explanations of the given observations. Independent of the way explanations are defined and computed, a general criterion which is suitable in this setting is the preference for least presumptive explanations [Poole, 1989], which generalize minimal (wrt set inclusion) explanations: an explanation that (also based on the IS-A hierarchy) implies another explanation is not least presumptive. In the following we assume that the candidates computed at each iteration represent the least presumptive explanations of the observations collected so far.

4 Choosing the Next Step

Let \( \Gamma \) be the current candidate set and let \( \omega = \{ O_1, \ldots, O_m \} \) encode the set of observations made so far. We assume that, if during some iteration we have observed a value \( O \) of a manifestation \( M \) and in a subsequent iteration we have observed a refinement \( o_k \) of \( O \), then \( O \) has been replaced by \( o_k \) in \( \omega \); therefore, for each manifestation \( M \), \( \omega \) contains the most specific value of \( M \) observed so far.

The set \( \omega \) encodes also the set of possible next observations to be performed, i.e. refining any non-leaf \( O \in \omega \). Therefore, in order to decide whether to stop or to proceed with a new observation, we select the minimum among:

- the action cost \( ac(\Gamma) \) associated with \( \Gamma \);
- for each non-leaf \( O \in \omega \), the estimated cost \( c(O) \), which is the sum of the cost \( oc(O) \) of refining \( O \) and the expected cost of the candidate set after refining \( O \), i.e.:

\[
    c(O) = oc(O) + \sum_{k=1}^{q} p(o_k|\Gamma) \cdot c(\Gamma_k)
\]

where \( \Gamma_1, \ldots, \Gamma_q \) are the possible candidate sets that would result by observing \( O \) and getting values \( o_1, \ldots, o_q \) respectively; \( p(o_k|\Gamma) \) is the probability of getting value \( o_k \) (computed based on current candidates \( \Gamma \), or the preferred candidates, as in [de Kleer and Williams, 1987]; de Kleer and Williams, 1989; Console et al., 1990]); and \( c(\Gamma_k) \) is the estimated cost of \( \Gamma_k \) as detailed in the following.

If \( ac(\Gamma) \) is the minimum among the costs, we stop; otherwise we observe the \( O \) with the smallest \( c(O) \); the rationale is that we proceed with a new observation only if we expect to be able to (eventually) achieve a reduction in the action cost worthy of the observation costs we expect to incur into.

Let \( \Gamma_k = \{ \gamma_1, \ldots, \gamma_s \} \) be one of the candidate sets involved in the above formula (note that each candidate \( \gamma_i \) may contain ground as well as abstract values of abducibles) and \( ac(\Gamma_k) \) be its action cost, i.e. the cost of stopping at \( \Gamma_k \), which must be compared with the estimated cost of acting after a further discrimination and refinement.

In principle, this estimation step would require to simulate all the possible observation sequences and outcomes and, for each of them, to assess the point where it is convenient, on average, to stop and perform the actions; in order to avoid such an intractable search, for the purpose of estimating \( c(\Gamma_k) \), we assume that the abductive process will continue as follows: first, one of the \( \gamma_i \in \Gamma_k \) is isolated; then, \( \gamma_i \) is refined by level, up to a point where performing an action is estimated to be convenient.

The rationale behind this heuristics is that the candidates in the set usually represent explanations that differ significantly from each other, and can therefore be discriminated (relatively cheaply) at the level of abstraction they are expressed into; only then, a more fine-grained tradeoff between acting and refining becomes worthwhile.

According to the discussion above, the estimated cost of \( \Gamma_k \) is defined as:

\[
    c(\Gamma_k) = \min(ac(\Gamma_k), ic(\Gamma_k) + rac(\Gamma_k)) \tag{3}
\]

where \( ic(\Gamma_k) \) is the estimated cost of isolating a single \( \gamma_i \in \Gamma_k \) and \( rac(\Gamma_k) \) is the estimated additional refinement and action cost once some \( \gamma_i \) has been isolated.

In this proposal, we estimate the cost \( ic(\Gamma_k) \) as follows:

\[
    ic(\Gamma_k) = \sum_{i=1}^{n} -p(\gamma_i|\Gamma_k) \cdot \log(p(\gamma_i|\Gamma_k)) \cdot \overline{ac}(\gamma_i) \tag{4}
\]

where \( -\log(p(\gamma_i|\Gamma_k)) \) is the estimated number of observations needed for isolating \( \gamma_i \) and \( \overline{ac}(\gamma_i) \) is an estimate of the cost of a single observation.

It is worth noting that, if we assume a model without hierarchical information (i.e. each \( \gamma_i \) is a ground candidate) and action costs are ignored (i.e. the goal is the identification of a single ground candidate), the expected cost \( c(\Gamma_k) \) in equation (3) is equal to \( ic(\Gamma_k) \). In such a case, if also observation costs are ignored (i.e. \( \overline{ac}(\gamma_i) = 1 \)), the formula above for \( ic(\Gamma_k) \) becomes the well-known formula proposed for iterative diagnosis in [de Kleer and Williams, 1987].

We have defined the estimate \( \overline{ac} \) of the cost of a single observation as a function of \( \gamma_i \), to possibly take into account the level of detail of observations related with \( \gamma_i \). In particular, in the domain knowledge \( K \), candidate \( \gamma_i \) may be related just with a subset of the manifestations and, for any such manifestation \( M \), it may be related just with a small number of refinements of values in the hierarchy \( \Lambda(M) \); therefore, the cost of an observation for isolating \( \gamma_i \) may be estimated by the average computed just on the costs of the relevant observations.

For estimating the cost \( rac(\gamma_i) \) of refining candidate \( \gamma_i = \{ H_{i,1}, \ldots, H_{i,r} \} \) until an action is taken, we assume that each of the abducible values \( H_{i,j} \) that compose \( \gamma_i \) is refined independently:

\[
    rac(\gamma_i) = \sum_{j=1}^{r} c(H_{i,j})
\]

where \( c(H_{i,j}) \) is the estimated cost associated with \( H_{i,j} \); then, costs \( rac(\gamma_i) \) are combined with a simple weighted sum into \( rac(\Gamma_k) \):
\[ \text{rac}(\Gamma_k) = \sum_{i=1}^{n} p(\gamma_i | \Gamma_k) \cdot \text{rac}(\gamma_i) \] (5)

The heuristics for computing \( \text{rac}(\gamma_i) \) is based on the fact that, according to our model, the actions associated with the values of an abducible \( A_i \) are disjoint from the actions associated with the other abducibles; therefore the appropriate level of refinement for the value \( H_{i,j} \) of abducible \( A_i \) does not depend on the other abducible values that appear in \( \gamma_i \).

In case action costs do not depend on the current context (as assumed in our model), each cost \( c(H_{i,j}) \) can be pre-computed offline. In particular, in this proposal we adopt a formula similar to the one for \( \text{rac}(\gamma_i) \), i.e.:

\[ c(H_{i,j}) = \min (ac(H_{i,j}), ic(H_{i,j}) + \text{rac}(H_{i,j})) \]

where \( ic(H_{i,j}) \) is the estimated cost of isolating a single child of \( H_{i,j} \) in hierarchy \( \Lambda(A_i) \) and \( \text{rac}(H_{i,j}) \) is the estimated additional refinement and action cost once some child of \( H_{i,j} \) has been isolated.

Cost \( ic(H_{i,j}) \) can be estimated with a formula similar to the one for \( \text{ic}(\gamma_i) \):

\[ ic(H_{i,j}) = \sum_{l=1}^{q} p(h_l | H_{i,j}) \cdot \log(p(h_l | H_{i,j})) \cdot \text{roc}(H_{i,j}) \]

where \( h_1, \ldots, h_q \) are the children of \( H_{i,j} \) in hierarchy \( \Lambda(A_i) \).

\( \text{roc}(H_{i,j}) \) is the estimated number of observations needed for isolating \( h_l \) and \( \text{roc}(H_{i,j}) \) is an estimate of the cost of a single observation. Considerations similar to the ones made for estimate \( \text{oc}(\gamma_i) \) apply to estimate \( \text{roc}(H_{i,j}) \).

The definition of \( \text{c}(H_{i,j}) \) is clearly recursive, since the computation of \( \text{rac}(H_{i,j}) \) involves the weighted sum of the estimated costs \( c(\cdot) \) of the children \( h_1, \ldots, h_q \) of \( H_{i,j} \):

\[ \text{rac}(H_{i,j}) = \sum_{l=1}^{q} p(h_l | H_{i,j}) \cdot c(h_l) \]

The recursion stops when \( H_{i,j} \) is a leaf of the hierarchy: in such a case, \( ic(H_{i,j}) = 0 \) (i.e. we do not need further discrimination) and \( \text{rac}(H_{i,j}) \) is the action cost \( ac(H_{i,j}) \) associated with leaf \( H_{i,j} \) in the model. This allows us to pre-compute \( c(H) \) for each possible value \( H \) of an abducible \( A \) with a bottom-up visit of \( \Lambda(A) \).

**Example**

Let us consider the execution of the explanation algorithm on the example in Figure 1.

**Precomputation step.** Since we assume that the actions do not depend on the context, the estimated costs \( c(H_{i,j}) \) associated with the abducibles can be pre-computed offline. For the leaves \( H_{i,j} \) of the hierarchies of abducibles, \( c(H_{i,j}) = ac(H_{i,j}) \). For the internal nodes, the estimated costs can be precomputed as shown in Table 1. For computing \( c(\text{Disease1.2}) \), e.g., we take into account the children \( \text{Disease1.2.1} \) and \( \text{Disease1.2.2} \) of \( \text{Disease1.2} \), their conditional probabilities \( p(\text{Disease1.2.1}|\text{Disease1.2}) = p(\text{Disease1.2.2}|\text{Disease1.2}) \) = \( \frac{1}{2} \), and the observation \( \text{LabTest1Pos} \) related with \( \text{Disease1.2} \). Therefore, \( c(\text{Disease1.2}) = \min (ac(\text{Disease1.2}), ic(\text{Disease1.2}) + \text{rac}(\text{Disease1.2})) = 6 \).

**Initial observations.** Let us suppose that an initial manifestation of \( \text{Symptom1} \) is detected, i.e., \( \text{Symptom1} \in \omega_1 \). The initial candidate set is \( \Gamma = \{\{\text{Disease1}\}, \{\text{Disease2}\}, \{\text{Disease3}\}\} \), representing the possible alternative diagnoses (in fact, \( \text{Disease1} \), \( \text{Disease2} \) and \( \text{Disease3} \) explain \( \text{Symptom1} \)).

**First iteration.** For choosing the next step, the algorithm evaluates whether to stop the diagnostic process and perform the treatment actions associated with \( \text{Disease1} \), \( \text{Disease2} \) and \( \text{Disease3} \) (which have cost \( ac(\Gamma) = ac(\text{Disease1}) + ac(\text{Disease2}) + ac(\text{Disease3}) = 35 \), or to perform a further observation, by taking into account their estimated costs. The possible observations which can be performed are \( \text{LabTest1} \) and \( \text{LabTest2} \), and the algorithm evaluates the cost \( c(O) \) of each of them.

Regarding the observation \( O = \text{LabTest1} \), two outcomes are possible: the test is either negative (\( \text{LabTest1Neg} \)) or positive (\( \text{LabTest1Pos} \)). For evaluating the candidate sets \( \Gamma_1 \) and \( \Gamma_2 \) resulting from the observation of \( \text{LabTest1} \), we adopt an approach similar to [Console et al., 1990]. In particular, if the outcome is \( \text{LabTest1Neg} \), it will be possible to exclude \( \text{Disease1.2} \), because this abducible explains \( \text{LabTest1Pos} \), which is incompatible with \( \text{LabTest1Neg} \); therefore \( \Gamma_1 = \{\{\text{Disease1}\}, \{\text{Disease2}\}, \{\text{Disease3}\}\} \). On the other hand, if the outcome is \( \text{LabTest1Pos} \), the minimal candidate set \( \Gamma_2 \) will be composed by \( \{\text{Disease1.2}\} \) only, which, since it explains \( \text{LabTest1Pos} \), will be part of every candidate set (see [Console et al., 1990]).

Regarding the observation \( O = \text{LabTest2} \), if the outcome of this observation is negative (\( \text{LabTest2Neg} \)), then \( \Gamma_1' = \{\{\text{Disease3}\}\} \). On the other hand, if the outcome is positive (\( \text{LabTest2Pos} \)), \( \Gamma_2' = \{\{\text{Disease1}\}, \{\text{Disease2}\}\} \).

In Tables 2 and 3 we report a summary of the estimation of the costs of the two observations, based on the estimated costs \( ic \) of isolating a single candidate, the estimated costs \( rac \) of refining the candidates, and the costs \( ac \) of directly treating the candidate diagnosis. For example, the estimated cost \( c(\Gamma_2 = \{\{\text{Disease1.2}\}\}) \) related with \( \Gamma_2 \) is computed as \( min (ac(\text{Disease1.2}), ic(\{\{\text{Disease1.2}\}\}) + rac(\{\{\text{Disease1.2}\}\})) \). The isolation cost \( ic(\{\{\text{Disease1.2}\}\}) \) is 0, because the candidate set \( \{\{\text{Disease1.2}\}\} \) is a singleton; \( rac(\{\{\text{Disease1.2}\}\}) = 0 \). In Table 2, \( \text{LabTest1Pos} \) and \( \text{LabTest1Neg} \) have been computed offline (see Table 1). Therefore, \( c(\Gamma_2 = \{\{\text{Disease1.2}\}\}) = 6 \).

Finally, the estimated cost of performing the observation \( \text{LabTest1} \) is computed on the base of the observation cost of \( \text{LabTest1} \) and the estimated costs associated with the outcomes \( \text{LabTest1Neg} \) and \( \text{LabTest1Pos} \): \( c(\text{LabTest1}) = ac(\text{LabTest1}) + p(\text{\text{LabTest1Neg}\mid\Gamma_1})c(\Gamma_1) + p(\text{\text{LabTest1Pos}\mid\Gamma_1})c(\Gamma_1') = 17.715 \).

This cost must be compared with the cost of performing the observation \( \text{LabTest2} \), i.e., \( c(\text{LabTest2}) = ac(\text{LabTest2}) + p(\text{\text{LabTest2Neg}\mid\Gamma_1})c(\Gamma_1) + p(\text{\text{LabTest2Pos}\mid\Gamma_1})c(\Gamma_1') = 20.991 \).
Table 1: Precomputed estimated costs of internal nodes of the abducible hierarchies. $H_{i,j}$ represents the internal node, $ic(H_{i,j})$ its estimated isolation cost, $rac(H_{i,j})$ its estimated refinement and action cost, and $c(H_{i,j})$ its total estimated cost.

<table>
<thead>
<tr>
<th>$H_{i,j}$</th>
<th>$ic(H_{i,j})$</th>
<th>$rac(H_{i,j})$</th>
<th>$c(H_{i,j})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disease1.2</td>
<td>$-\frac{1}{2} \log(\frac{1}{2})oc(LabTest1Pos) - \frac{1}{2} \log(\frac{1}{2})oc(LabTest1Pos) = 8$</td>
<td>$\frac{1}{2}(\text{ac}(Disease1.1.1) + \text{ac}(Disease1.2.2)) = 4$</td>
<td>6</td>
</tr>
<tr>
<td>Disease1</td>
<td>$-\frac{1}{2} \log(\frac{1}{2})oc(LabTest2) - \frac{1}{2} \log(\frac{1}{2})oc(LabTest1Pos) = 7$</td>
<td>$\frac{1}{2}(\text{ac}(Disease1.1) + \text{ac}(Disease1.2)) = 7$</td>
<td>10</td>
</tr>
<tr>
<td>Disease2</td>
<td>$-\frac{1}{2} \log(\frac{1}{2})oc(LabTest2Pos) - \frac{1}{2} \log(\frac{1}{2})oc(LabTestPos) = 10$</td>
<td>$\frac{1}{2}(\text{ac}(Disease2.1) + \text{ac}(Disease2.2)) = 7$</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2: Possible observations at the first iteration of the explanation algorithm. $O$ represents a possible observation outcome, $\Gamma_k$ the related candidate set, and $p(o_k|\Gamma)$ the conditional probability of the outcome.

| $O$ | $o_k$ | $\Gamma_k$ | $p(o_k|\Gamma)$ |
|-----|-------|-------------|-----------------|
| LabTest1 | $o_1 = \text{LabTest1Neg}$ | $\Gamma_1 = \{\{Disease1.1\}, \{Disease2\}, \{Disease3\}\}$ | $p(\text{LabTest1Neg}|\Gamma) = \frac{\text{ic}(\Gamma_1)}{\text{rac}(\Gamma_1)} = \frac{1}{2}$ |
| LabTest1 | $o_2 = \text{LabTest1Pos}$ | $\Gamma_2 = \{\{Disease1.2\}\}$ | $p(\text{LabTest1Pos}|\Gamma) = \frac{\text{ic}(\Gamma_2)}{\text{rac}(\Gamma_2)} = \frac{1}{2}$ |
| LabTest2 | $o_1' = \text{LabTest2Neg}$ | $\Gamma_1' = \{\{Disease3\}\}$ | $p(\text{LabTest2Neg}|\Gamma) = \frac{\text{ic}(\Gamma_1')}{\text{rac}(\Gamma_1')} = \frac{3}{2}$ |
| LabTest2 | $o_2' = \text{LabTest2Pos}$ | $\Gamma_2' = \{\{Disease1\}, \{Disease2\}\}$ | $p(\text{LabTest2Pos}|\Gamma) = \frac{\text{ic}(\Gamma_2')}{\text{rac}(\Gamma_2')} = \frac{1}{2}$ |

Table 3: Estimated costs at the first iteration of the explanation algorithm. $\Gamma_k$ represents a candidate set, $ic(\Gamma_k)$ its estimated isolation cost, $rac(\Gamma_k)$ its estimated refinement and action cost, and $c(\Gamma_k)$ its total estimated cost.

| $O$ | $o_k$ | $\Gamma_k$ | $p(o_k|\Gamma)$ |
|-----|-------|-------------|-----------------|
| LabTest1Pos | $o_1'' = \text{LabTest1+}$ | $\Gamma_1'' = \{\{Disease1.2.1\}\}$ | $p(\text{LabTest1+}|\Gamma) = \frac{\text{ic}(\Gamma_1'')}{\text{rac}(\Gamma_1'')} = \frac{1}{2}$ |
| LabTest1Pos | $o_2'' = \text{LabTest1++}$ | $\Gamma_2'' = \{\{Disease1.2.2\}\}$ | $p(\text{LabTest1++}|\Gamma) = \frac{\text{ic}(\Gamma_2'')}{\text{rac}(\Gamma_2'')} = \frac{1}{2}$ |
| LabTest2 | $o_1''' = \text{LabTest2Neg}$ | $\Gamma_1''' = \emptyset$ | 0 |
| LabTest2 | $o_2''' = \text{LabTest2Pos}$ | $\Gamma_2''' = \{\{Disease1.2\}\}$ | 1 |

Table 4: Possible observations at the second iteration of the explanation algorithm, after observing $\text{LabTest1Pos}$.

<table>
<thead>
<tr>
<th>$\Gamma_k$</th>
<th>ic($\Gamma_k$)</th>
<th>rac($\Gamma_k$)</th>
<th>c($\Gamma_k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma_1$</td>
<td>0</td>
<td>$1 \cdot c(Disease1.2.1) = 4$</td>
<td>$\text{min}(\text{ac}(Disease1.2.1), \text{ic}(\Gamma_1) + \text{rac}(\Gamma_1)) = 4$</td>
</tr>
<tr>
<td>$\Gamma_2$</td>
<td>0</td>
<td>$1 \cdot c(Disease1.2.2) = 4$</td>
<td>$\text{min}(\text{ac}(Disease1.2.2), \text{ic}(\Gamma_2) + \text{rac}(\Gamma_2)) = 4$</td>
</tr>
<tr>
<td>$\Gamma_3$</td>
<td>0</td>
<td>$1 \cdot c(Disease1.2) = 6$</td>
<td>$\text{min}(\text{ac}(Disease1.2), \text{ic}(\Gamma_3) + \text{rac}(\Gamma_3)) = 6$</td>
</tr>
</tbody>
</table>

Table 5: Estimated costs at the second iteration of the explanation algorithm.

$c(\text{LabTest1})$ and $c(\text{LabTest2})$, and it is less than the cost $ac(\Gamma)$ of treating both $\text{Disease1}$ and $\text{Disease2}$, the algorithm chooses to perform $\text{LabTest1}$ and to observe its outcome.

**Second iteration.** Let us suppose that the outcome of $\text{LabTest1}$ is positive (i.e., $\text{LabTest1Pos}$). Then, the algorithm performs a further iteration with $\Gamma = \Gamma_2 = \{\{\text{Disease1.2}\}\}$, in order to choose whether to stop, or to refine the observation $\text{LabTest1Pos}$, or to perform the observation $\text{LabTest2}$.

Let us consider $O = \text{LabTest1Pos}$. The possible outcomes are $\text{LabTest1+}$ and $\text{LabTest1++}$. If the outcome is $\text{LabTest1+}$, the candidate set $\Gamma_1''$ is $\{\{\text{Disease1.2.1}\}\}$...
if the outcome is \textit{LabTest1++}, the candidate set \( \Gamma''_2 \) is \( \{ \text{Disease1.2.2} \} \) (see Figure 1).

As reported in Tables 4 and 5, the estimated cost of refining the observation \textit{LabTest1Pos} can be computed as \( c(\text{LabTest1Pos}) = oc(\text{LabTest1Pos}) + p(\text{LabTest1})c(\Gamma''_1) + p(\text{LabTest1++})c(\Gamma''_2) = 8 + \frac{1}{4} + \frac{1}{4} = 12 \).

Let us consider \( O = \text{LabTest2} \). The possible outcomes are \textit{LabTest2Neg} and \textit{LabTest2Pos}. If the outcome is \textit{LabTest2Neg}, the candidate set \( \Gamma''_2 \) will be empty, because the only (minimal) candidate diagnosis \textit{Disease1.2} is not compatible with this outcome. If the outcome is \textit{LabTest2Pos}, the candidate set \( \Gamma''_2 \) will be \( \{ \text{Disease1.2} \} \). Therefore, \( c(\text{LabTest2}) = oc(\text{LabTest2}) + 1 \cdot c(\Gamma''_2) = 14 \).

Since \( ac(\Gamma) = 6 \) is less than both the estimated costs of the possible observations \( c(\text{LabTest1Pos}) \) and \( c(\text{LabTest2}) \), the algorithm stops here and it does not perform further observations. Therefore, in this example, the best option is to not continue to request further laboratory examinations for refining the diagnosis, but to treat the patient for \textit{Disease1.2}.

5 Allowing Multiple Instances

In the discussion made so far we have assumed that at most one of the leaf values in a hierarchy (either \( \Lambda(A) \) or \( \Lambda(M) \)) is true in each situation. In particular, this implies that, for each abducible \( A_j \), a candidate \( \gamma_i \) can contain at most one value \( H_{i,j} \) drawn from the hierarchy \( \Lambda(A_j) \); moreover, a candidate \( \gamma_i \) (either mentioning multiple values of some abducible \( A_j \) or not), should not imply two different values for the same manifestation \( M \).

In this section we want to discuss the implications of relaxing this assumption, i.e. of considering the possibility that some observations result in two different instances of a manifestation \( M \) and/or that some of these observations can only be explained by assuming the presence of two or more instances of \( A_j \).

An example of this kind of situations may be a model where the abducibles are viral infections, and the observations are the symptoms. We may have an abducible \( VI \) associated with a hierarchy \( \Lambda(VI) \) of viral infections and a manifestation \( SYM \) associated with a hierarchy \( \Lambda(SYM) \) of symptoms; we can imagine situations when the patient has contracted two different infections from hierarchy \( \Lambda(VI) \), so that two different symptoms from the hierarchy \( \Lambda(SYM) \) are observed, and they can only be explained by assuming two infections from the hierarchy \( \Lambda(VI) \).

Let us first consider the outcome of an observation which refines a value \( O \) of a manifestation \( M \). Contrary to our previous discussion, the outcome of the observation may include any subset of one or more of the children \( \text{child}(O) = \{ o_1, \ldots, o_q \} \) of \( O \) in the hierarchy \( \Lambda(M) \); let us define the set \( \Omega(O, \Gamma) \) as:

\[
\Omega(O, \Gamma) = \{ \omega_O \in \mathcal{P}(\text{child}(O)) : p(\omega_O | \Gamma) \neq 0 \}
\]

where \( \mathcal{P}(\text{child}(O)) \) denotes the powerset of \( \text{child}(O) \); the set \( \Omega(O, \Gamma) \) contains all the combinations \( \omega_O \) of one or more of the children of \( O \) whose probabilities are strictly positive given that the candidate set is \( \Gamma \).

Then, the formula for computing the cost of \( O \) (i.e. equation (2)) should be generalized to:

\[
c(O) = oc(O) + \sum_{\omega_O \in \Omega(O, \Gamma)} p(\omega_O | \Gamma) \cdot c(\Gamma_{\omega_O})
\]

where \( \Gamma_{\omega_O} \) is the candidate set that would result by observing \( O \) and getting the set of values \( \omega_O \). Note that, if we assume \( p(\omega_O | \Gamma) = 0 \) for each \( \omega_O \) which does not contain exactly one of the children of \( O \), we get back equation (2).

It should be noted that, at least given some candidate sets \( \Gamma \), the a-posteriori probability of combinations \( \omega_O \) containing more than one value may be relatively high (the extreme case being that given the current candidate set \( \Gamma \) only non-singleton combinations have a positive probability).

Let us now consider the computation of the probability of a value \( h_{i,j} \) in a hierarchy \( \Lambda(A_j) \), needed in equations (4) and (5) for computing \( p(\gamma_i | \Gamma_k) \); indeed, by assuming that the abducibles are independent, if \( \gamma_i = \{ H_{i,1}, \ldots, H_{i,r_i} \} \) we have that:

\[
p(\gamma_i | \Gamma_k) = \frac{p(\gamma_i)}{p(\Gamma_k)} = \left( \prod_{k=1}^{r_i} p(H_{i,k}) \right) / p(\Gamma_k)
\]

Under our previous assumption that, if \( H_{i,j} \) is true, exactly one of the children \( h_1, \ldots, h_{q} \) of \( H_{i,j} \) is true, we have that:

\[
p(H_{i,j}) = \sum_{k=1}^{q} p(h_k)
\]

Then, the values \( p(h_k) \) are recursively computed by applying the same formula, until we reach the leaves of the hierarchy, whose probabilities are given with the model.

When the mutual exclusion assumption is relaxed, we should consider that also combinations of two or more children can be true. If we let \( \Gamma(H_{i,j}) \) be the powerset \( \mathcal{P}(\text{child}(H_{i,j})) \) of the set of children of \( H_{i,j} \) (excluding the empty set \( \emptyset \)), we have:

\[
p(H_{i,j}) = \sum_{\gamma_{H_{i,j}} \in \Gamma(H_{i,j})} p(\gamma_{H_{i,j}})
\]

where the probability of each combination \( \gamma_{H_{i,j}} \) is obtained by multiplying the probabilities \( p(h_k) \) of the children \( h_k \) that appear in \( \gamma_{H_{i,j}} \), and the negated probabilities \( (1 - p(h_k)) \) of the children \( h_k \) that do not appear in \( \gamma_{H_{i,j}} \).

However, if we assume that the a-priori probabilities given for the leaves of the hierarchies of abducibles are very low (as it happens, e.g., when they represent diseases or faults), the simpler formula that assumes mutual exclusion can be a very good approximation of the more general (but computationally expensive) formula. Indeed, in such a case the total probability that two or more children are true may be so low that it can be safely ignored.

Similar considerations apply to the computation of the probabilities of candidates \( \gamma_i \) (and therefore, indirectly, to the computation of the probabilities of observing a combination \( \omega_O \) of a value \( O \)). Since each candidate \( \gamma_i \) is least presumptive, it only mentions the values of abducibles that must be true in order for \( \gamma_i \) to explain the observations collected so far. In this way, two candidates \( \gamma_i \) and \( \gamma_j \) in a candidate set \( \Gamma_k \)

Alban Graußen, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
are not necessarily mutually exclusive, since they may have common extensions: if, e.g. $\gamma_i = \{A_1\}$ and $\gamma_j = \{A_2\}$, both candidates share the explanation $\{A_1, A_2\}$. However, for the same reasons discussed above, ignoring the probability of $\{A_1, A_2\}$ is usually safe, provided the a-priori probabilities of the leaves of the hierarchies are low. In such a case, we can use the formulas developed in section 4 as good approximations.

6 Conclusions

In this paper we proposed an approach to selecting the next step in an abductive explanation loop which extends previous work on measurement selection in Model-Based Reasoning. In fact, it is based on a representation with abstractions, and, depending on the costs of observations and the costs of actions to be taken, a further observation may be chosen for discriminating or refining current candidates, or the loop can be terminated, so that actions will be taken based on the current candidate(s). Costs of observations and actions may be very different at different levels of abstraction.

The approach is aimed at being general, because its motivations can be found in several tasks and domains including technical and medical diagnosis as well as interpretation tasks. Different instances may be derived with specific approaches for representing domain knowledge and for generating and updating candidate explanations based on observations.

The complexity of the proposed algorithm is obviously strongly influenced by such choices and, in particular, by the complexity of the algorithms for generating and updating explanations as well as by the number of candidate explanations to be considered at each iteration. Given the similarities outlined before, most of the complexity analysis of the selection of the next observation would be identical to those of the proposals of [de Kleer and Williams, 1987] and [Console et al., 1990], except for the specifics needed to handle the hierarchies and costs which, however, cannot be the cause of combinatorial explosion; on the other hand, by exploiting hierarchies, we expect on the average smaller sets of explanations to be inspected for choosing the next step.

References


A Heuristic for Near-Optimal Troubleshooting Using AO*

Håkan Warnquist
 Dept. of Computer and Information Science
 Linköping University
 SE-581 83 Linköping, Sweden
 g-hakwa@ida.liu.se

Mattias Nyberg
 Dept. of Electrical Engineering
 Linköping University
 SE-581 83 Linköping, Sweden
 mattias.nyberg@scania.com

Abstract

When troubleshooting malfunctioning technical equipment, the task is to locate faults and make repairs until the equipment functions properly again. The AO* algorithm can be used to find troubleshooting strategies that are optimal in the sense that the expected cost of repair is minimal. We have adapted the AO* algorithm for troubleshooting in the automotive domain with limited time. We propose a new heuristic based on entropy. By using this heuristic, near-optimal strategies can be found within a fixed time limit. This is shown in empirical studies on a fuel injection system of a truck. In these results, the AO* algorithm using the new heuristic, performs better than other troubleshooting algorithms.

1 Introduction

The task of a troubleshooter is to locate faults and make repairs in machinery and technical equipment. For example, in the automotive domain, the troubleshooter is a mechanic that resolves a problem on a vehicle by repairing components. To know which components to repair, the mechanic can make observations to locate the cause of problem. As vehicles get more complex, the troubleshooting task becomes more difficult. An automated troubleshooter infers probabilities of component faults given observations and aids the mechanic by recommending appropriate actions to perform.

If we also want to minimize the costs, the troubleshooting task becomes an optimization problem that can be solved as a planning problem. This problem is known to be NP-hard so if we want the computations to be made while the mechanic is waiting, approximate methods are needed [Vomlelová and Vomlel, 2000].

AO* is a well known heuristic search algorithm that can solve planning problems involving uncertainty and feedback. It uses a heuristic function to focus its search. Provided that the heuristic is admissible the solution found is optimal [Nilsson, 1980]. There are heuristics that have been used with AO* to solve the troubleshooting problem in the literature [Faure, 2001; Vomlelová and Vomlel, 2000; Raghavan et al., 1999], but we need stronger heuristics to focus the search more to find solutions of larger problems.

In this paper we propose a way to use AO* to find near-optimal solutions in a limited time. We use a new heuristic that, in empirical experiments on a case study, is shown to find solutions with lower costs compared to heuristics used in [Raghavan et al., 1999; Vomlelová and Vomlel, 2000]. The case study is from the automotive domain and it is the fuel injection system of a truck.

In Section 2 we give the problem formulation and in Section 3 we present the case study. In Section 4 we describe some solution methods from the literature to the troubleshooting problem and other related problems. In Section 5 we show how AO* is used to solve the troubleshooting problem. In Section 6 we present the new heuristic. In Section 7 we do an empirical evaluation of the implementation and the new heuristic and finally we conclude in Section 8.

2 Problem Formulation

We will use the problem formulations of the troubleshooting frameworks presented in [Heckerman et al., 1995], [Vomlelová and Vomlel, 2000], and [Langseth and Jensen, 2002]. In the troubleshooting problem the probabilistic relationships between component faults and observations is represented as a Bayesian network [Jensen, 1996]. With this probabilistic model it is possible to infer probability distributions of component faults conditioned on the information gained from previously performed actions.

Actions that we can perform on the system are either repair actions that successfully repair a single component or observing actions that observe the value of a node in the Bayesian network. Each action a is associated with a cost ca. This cost is independent of any previously performed actions. In the Bayesian network there exists a single problem-defining node that indicates if any component is non-functioning. The observing action that observes this node is called the function control. The troubleshooting is said to be successful when a successful function control is made and the problem-defining node is observed to be non-indicating.

A troubleshooting strategy is defined as a labeled directed tree that describes the process of performing actions until the process terminates. The edges of the directed tree are labeled with actions. Edges labeled with observing actions are associated with a specific outcome of the action. Branching occurs only from nodes where each outgoing edge is labeled with an observing action. An example of a troubleshooting strategy is shown in Figure 1.
When a troubleshooting strategy is executed, we perform actions on the path from the root to a terminal node resulting from the responses of the observing actions. A terminal node is said to be successful if the action on the last edge is a successful function control and a successful troubleshooting strategy is a troubleshooting strategy where all terminal nodes are successful. This is done to validate that the problem is resolved.

The cost of reaching a terminal node \( t \) from the root node of a troubleshooting strategy \( s \), \( CR(s, t) \), is

\[
CR(s, t) = \sum_{a \in \mathcal{P}(s, t)} c_a
\]

where \( \mathcal{P}(s, t) \) is the set of all actions on the path from the root of the strategy \( s \) to the node \( t \) and \( c_a \) is the cost of the action \( a \).

Let \( T(s) \) be the set of all terminal nodes of the troubleshooting strategy \( s \) and let \( c \) be the current evidence representing our accumulated knowledge of the system, i.e. the results of all previously performed actions. When \( s \) is executed a terminal node \( t \in T(s) \) is reached with a certain probability, \( P(T = t | s, c) \), where \( T \) is a stochastic variable with the outcome space \( T(s) \). These probabilities can be obtained from the probabilistic model. The expected cost of repair of \( s \) given \( c \), \( ECR(s, c) \), is the expectation of the cost of reaching any node in \( T(s) \):

\[
ECR(s, c) = E(CR(s, T) | s, c) = \sum_{t \in T(s)} P(T = t | s, c) CR(s, t)
\]

The task in the troubleshooting problem is to find a successful troubleshooting strategy \( s^* \) that is optimal in the sense that it minimizes the expected cost of repair given the current evidence \( c \):

\[
s^* = \arg\min_s ECR(s, c)
\]

### 3 Case Study — A Fuel Injection System

We will use an existing fuel injection system of a truck as a case study when evaluating the here proposed algorithms. This is a motivating and inspiring example from the real world since it is particularly hard for the on-board diagnosis system to isolate component faults.

The fuel injection system uses a high injection pressure to inject diesel in the cylinders of the engine. For many mechanical faults on this system, the only symptom is a loss in injection pressure. These faults can only be located by manual testing. A picture of the fuel injection system is shown in Figure 2.

![Figure 2: Extreme High Pressure Fuel Injection System.](image)

The system is modeled with 17 components (pipes, pumps, filters, and valves) of which 7 are observable, i.e., a single observing action can unambiguously determine if the component is faulty. In total there are 30 binary observations that can be made (e.g., cylinder balancing, visible leakage, and air in fuel test). 21 of them can be accessed by observing actions. The remaining 9 observations are generated by the on-board diagnosis system of the truck and their values are given prior to the troubleshooting.

The dependencies between component faults and observations are modeled with a Bayesian network. Details on how this information was retrieved can be found in [Mossberg, 2007]. Apart from the assumption that at most one component can be faulty at the same time there are no causal dependencies between the faults. Observations are dependent on current component faults and on themselves in the sense that an observing action will always yield the same result when repeated unless any of the faults, that the observation is causally dependent on, is repaired.

The costs of the actions are heterogeneous, but in general the repair actions are more expensive than observing actions. The function control is the most expensive observing action since it requires a test drive of the vehicle.

### 4 Approximate and Exact Solutions to the Troubleshooting Problem in the Literature

In this section we will describe some different approaches to the troubleshooting problem. There are the greedy approaches mainly based on work by [Heckerman et al., 1995] and [Langseth and Jensen, 2002] where time efficient algorithms are used to find approximate solutions to the troubleshooting problem. There are also various search based methods where more accurate solutions are found by searching at a higher cost in time.

#### 4.1 Solving the Troubleshooting Problem Using Greedy Algorithms

In [Heckerman et al., 1995] a special case of the troubleshooting problem is solved optimally in linear time using a greedy
algorithm. We will take some time to describe this approach since we will compare our approach with this one in Section 7.3. The problem formulation of the basic troubleshooting problem is extended with the following assumptions:

- There can only be at most one faulty component.
- Immediately following any component repair a function control must be made.
- When the troubleshooting starts the problem-defining node is observed to be indicating, i.e. we know that exactly one component is faulty.
- Some components are observable, i.e. an observing action can be made that unambiguously determines if the component is functioning or not.
- No other observing actions are available.

Under these assumptions it is proved that an optimal troubleshooting strategy that minimizes the expected cost of repair can be obtained by always performing the action corresponding to the component with the highest efficiency first. The efficiency of an observable component is defined as the ratio of the probability that the component is faulty and the cost of observing it. For unobservable components the efficiency is defined as the ratio of the probability and the sum of the costs of repairing the component and making the function control.

Let $p_i = P(\text{component } i \text{ is faulty} | r)$. Then, using (2), the expected cost of repair using the strategy based on efficiencies $s_{\text{eff}}$ on a system with $n$ components is

$$ECC(s_{\text{eff}}, \epsilon) = \sum_{i=1}^{n} p_i \left( \sum_{j=1}^{i} c_{i,j} + c_{i,r} + c_{i,c} \right)$$

where $c_{i,r}$ is the cost of observing component $i$, $c_{i,r}$ is the cost of repairing component $i$ and $c_{i,c}$ is the cost of making the function control. An unobservable component is "observed" by first repairing it and then performing a function control. For these components $c_{i,r}$ is the cost of repairing the component and making the function control and $c_{i,c} = -c_{i,c}$ so that $c_{i,r}$ and $c_{i,c}$ cancel each other in (4).

If any of the assumptions above are relaxed $s_{\text{eff}}$ is no longer guaranteed to be optimal. However, near optimal troubleshooting strategies can be found by extending the strategy based on efficiencies with a two step look-ahead algorithm when more general observing actions also are available [Langseth and Jensen, 2002]. The two step look-ahead algorithm works by only allowing general observing actions to be either performed immediately in this time step or in the next.

Greedy algorithms can be constructed to solve the troubleshooting problem in other ways than using efficiencies. For example, in [de Kleer and Williams, 1987; Gillblad et al., 2006] the next action to be performed is chosen as the one that maximizes the information gain. However, in empirical tests troubleshooting printers, the two step look-ahead algorithm is shown to find near-optimal solutions with an error of only a few percent [Vomlelova and Vomlel, 2000; Langseth and Jensen, 2002].

### 4.2 Solving the Troubleshooting Problem by Searching

Solving the troubleshooting problem optimally is a planning problem involving uncertainty and feedback. These types of problems can in a natural way be formulated as AND/OR graphs in which an optimal solution can be found by searching [Ghallab et al., 2004; Bonet and Geffner, 2000]. AND/OR graphs are often used when solving the troubleshooting problem by searching. In [Vomlelova and Vomlel, 2000] they are used to describe and optimally solve small troubleshooting problems and in [Raghavan et al., 1999; Olive et al., 2003] they are used when finding optimal and near-optimal solutions to the test sequencing problem. The test sequencing problem is similar to the troubleshooting problem, where the goal is to isolate the fault by performing tests to progressively gain more information.

**AND/OR Graphs**

An AND/OR graph can be represented as a labeled directed hypergraph with a single root node [Nilsson, 1980]. The edges connect one parent node with one or more successor nodes. An edge connecting a parent node to $k$ successor nodes is called a $k$-connector. AND/OR graphs are sometimes represented as regular directed graphs where the nodes of the hypergraph are called OR nodes and $k$-connectors with $k \geq 2$ are replaced by nodes called AND nodes with one incoming edge and $k$ outgoing edges. An example of an AND/OR graph represented by a hypergraph is shown in Figure 3.

![Figure 3: An AND/OR graph represented as a hypergraph. 2-connectors are shown as arrows joined with arcs.](image-url)

When the AND/OR graph is used to describe the troubleshooting problem, each node represents a decision point where each outgoing connector represents an action that can be chosen to be performed. When a repair action is performed another decision point is reached. The nodes representing these decision points are connected by a 1-connector labeled with the repair action. When an observing action with $k$ possible outcomes is performed, different decision points will be reached depending on the outcome. These nodes are connected by a $k$-connector labeled with the observing action. A subgraph $s$ of an AND/OR graph $G$ is a solution of $G$ corresponding to a troubleshooting strategy if the following conditions are true [Vomlelova and Vomlel, 2000]:

- the root of $s$ is also the root of $G$. 

• if \( n \) is a non-terminal node in \( s \), then exactly one outgoing connector from \( n \) also belong to \( s \).
• if \( c \) is a connector in \( s \), then all successor nodes of \( c \) also belong to \( s \).
• all terminal nodes in \( s \) are leaf nodes in \( G \).

Since a solution corresponds to a troubleshooting strategy the cost of a solution can be defined by (2).

### Algorithms for Finding Solutions in AND/OR Graphs

Describing the troubleshooting problem like this allows us to use existing algorithms used for finding solutions in AND/OR graphs such as Value Iteration [Bertsekas, 1995] and AO* [Martelli and Montanari, 1978; Nilsson, 1980] to find optimal troubleshooting strategies.

Value Iteration is a general algorithm often used for finding optimal solutions in Markov Decision Processes. In [Casandra et al., 1998] Value Iteration is adapted to solve problems formulated as Partially Observable Markov Decision Processes which is a problem formulation similar to AND/OR graphs. In Value Iteration, estimates of the optimal solution are updated sequentially for every possible state of the problem until convergence. An advantage with this algorithm is that once a solution is found the optimal troubleshooting strategy is also found for every possible initial state. However, since an estimate needs to be assigned to every possible state Value Iteration becomes inefficient when the state space is large.

AO* is a search algorithm that makes use of a heuristic function to focus the search when finding optimal solutions in acyclic AND/OR graphs. The heuristic is used to estimate cost-to-go in each leaf node, i.e. the cost of an optimal solution of the subproblem rooted in the leaf node. When good heuristics are available AO* has been shown to perform better than Value Iteration for solving AND/OR graphs [Bonet and Geffner, 2005]. The AO* algorithms expand the AND/OR graph node by node and keep track of the currently best partial solution given heuristic estimates of the cost-to-go in each leaf. This gives AO* the advantage that it can be stopped prematurely returning a suboptimal partial solution.

A disadvantage with AO* is that it cannot handle cycles and that it is memory intensive. However, there are variants of AO* that handle cyclic AND/OR graphs [Hansen and Zilberstein, 2001], but it turns out that, for our problem, cycles in the AND/OR graph can be prevented by forbidding certain actions without loss of optimality. Also, there are memory bounded variants of AO* [Chakrabarti et al., 1989] so that the memory usage can be controlled by trading space with time.

### 5 Using AO* to Solve the Troubleshooting Problem

In this section we will describe how AO* is implemented to solve troubleshooting problem such as the one described in Section 3. First we will give a brief overview of how the AO* algorithm works as described in [Nilsson, 1980]. Then we show how we do the state representation of the troubleshooting problem and how we treat this in our implementation. We will also describe how we use AO* to search with limited time.

#### 5.1 Overview of the AO* Algorithm

Let \( G \) be the implicit AND/OR graph of the troubleshooting problem and let \( G' \) be the explicit subgraph of \( G \) consisting of the nodes and connectors that have been explored by the algorithm. Every node \( n \) in \( G' \) are associated with a cost \( g_n \) and labeled with a state \( S \in S \) that describes the system at a given moment where \( S \) is the state space. The cost of a leaf node \( n \) in \( G' \) that is not a leaf in \( G \) is given by a heuristic function \( h : S \rightarrow \mathbb{R} \). The best partial solution is an arbitrarily chosen solution of \( G' \) that minimizes (2). Let this solution be denoted \( \tilde{s} \).

The algorithm starts the search with \( \tilde{s} \) consisting solely of the root node of \( G \). Until \( \tilde{s} \) is also a solution of \( G' \), the node with the highest cost of the terminal nodes in \( \tilde{s} \) that are not leaves in \( G \) is expanded. Whenever a node is expanded, the currently best solution \( \tilde{s} \) is updated. The algorithm terminates once a solution of \( G' \) is found. If the heuristic function is admissible, i.e. it never over-estimates the true cost-to-go, the solution corresponds to the troubleshooting strategy that minimizes (2).

#### 5.2 State Representation

The state that we label each node with needs to contain the information necessary to describe our knowledge of the system at that point in the troubleshooting process. If we assume the Markov property, we can describe our knowledge with a belief state in accordance with [Russell and Norvig, 2003].

Let \( f_{i,k} : i \geq 1 \) be the event that component \( i \) is faulty at time \( k \) and let \( f_{0,k} \) be the event that the system has no faults at time \( k \). Let \( e_{1:k} \) be the accumulated evidence from time \( k \). If no new information is gained, the probability that component \( i \) is faulty remains the same at time \( k + 1 \):

\[
P(f_{i,k+1}|e_{1:k}) = P(f_{i,k}|e_{1:k})
\]

Let us now assume that observations are only dependent on current component faults. Then:

\[
P(e_k|f_{1:k}, e_{1:k-1}) = P(e_k|f_{1:k})
\]

where \( e_k \) is an observation made at time \( k \).

If we know the probability distributions of all faulty components given all accumulated evidence at time \( k - 1 \), we can calculate the probability distributions of all faulty components given the accumulated evidence at time \( k \).

\[
P(f_{i,k}|e_{1:k}) = P(f_{i,k}|e_k, e_{1:k-1}) \propto P(e_k|f_{1:k}, e_{1:k-1})P(f_{i,k}|e_{1:k-1})
\]

\[
(P(5)(6))
\]

This probability distribution is referred to as the belief state.

**Definition 1.** (Belief state) The belief state is a vector \( b_k \) containing the probability distribution over component faults given the accumulated evidence \( e_{1:k} \) at time \( k \). Each element \( b_k(i) \) is

\[
b_k(i) = P(f_{i,k}|e_{1:k})
\]

The belief state \( b_0 \) at time 0 is the à priori probability distribution over component faults.
When an observing action is performed the evidence $e_k$ is gained and we can calculate a new belief state $b_k$ from the previous belief state $b_{k-1}$. For each element $b_k(i)$ in $b_k$

$$b_k(i) = \begin{cases} \frac{1}{P(f_i | e_k)} & \text{(7)} \\ \propto P(e_k | f_i) P(f_{i,k-1} | e_{1:k-1}) & \text{(8)} \\ \propto P(e_k | f_i) b_{k-1}(i) & \text{(9)} \end{cases}$$

Since we assume single faults, when a repair action is performed we increase the probability that the system is free of faults with the probability that the repaired component was faulty and then we set that probability to zero. After component $j$ is repaired, for each element $b_k(i)$ in $b_k$

$$b_k(i) = \begin{cases} b_{k-1}(0) + b_{k-1}(j) & \text{if } i = 0 \\ 0 & \text{if } i = j \\ b_{k-1}(i) & \text{otherwise} \end{cases}$$

### 5.3 Repeated Observing Actions

For the fuel injection system described in Section 3 we said that an observing action that is repeated always yields the same result unless any of the faults, that the observation is causally dependent on, is repaired. This means that no new information can be gained by that observing action. Even though this violates the Markov property, we do not have to change the probabilistic model if we add a small exception to (9).

We propose to keep track of all recently made observing actions at time $k$ in a set $O_k$. When a component is repaired we remove, from the same set, all observing actions that observes a node that is causally dependent on this component fault. Let $A_i$ be the set of all observing actions observing nodes that are causally dependent on the component fault $i$. Then if component $i$ is repaired the set of all recently made observations is updated such as

$$O_k = O_{k-1} \setminus A_i$$

and when an observing action $o$ is performed it is updated such as

$$O_k = O_{k-1} \cup \{o\}$$

Let $e_k$ be the evidence gained from an observing action $o$. Then instead of using (9), after $o$ is performed we update the belief state such that for each element $b_k(i)$ in $b_k$

$$b_k(i) = \begin{cases} b_{k-1}(i) P(e_k | f_i) b_{k-1}(i) & \text{if } o \in O_{k-1} \\ b_{k-1}(i) & \text{otherwise} \end{cases}$$

### 5.4 Expanding Nodes

When the algorithm expands a node, given the state a limited amount of actions can be performed. A repair action yields a 1-connector connected to a node labeled with a state created from the previous state using (10) and (11). If no node labeled with that state already exists in the explicit graph $G'$, a new node is created. A binary observing action yields a 2-connector connected to nodes labeled with states created using (13) and (12).

When expanding, we will only consider actions that bring the system closer to the goal of repairing the system, i.e. repair actions that repair component faults with a probability greater than zero and observing actions from which new information can be gained. These actions are said to be applicable actions.

**Definition 3.** (Applicable Action) Let $B'$ be the set of resulting belief states when action $a$ is performed on the state $S = (b, O)$. An action $a$ is applicable in $S$ if there exists $b' \in B'$ such that $b' \neq b$.

**Remark.** When a binary observing action is performed, $B'$ consists of two belief states, one for each possible outcome of the observation. When a repair action is performed $B'$ consists a single belief state.

### 5.5 Updating the Best Partial Solution

Each connector is associated with an action $a$ and a probability $p_m$ for each successor node $m$. For a 2-connector $p_m$ is the probability of having the corresponding outcome $e_k$ given the accumulated evidence $e_{1:k-1}$:

$$p_m = P(e_k | e_{1:k-1})$$

$$S = \sum_i P(e_k | f_{i,k}) P(f_{i,k} | e_{1:k-1})$$

$$S = \sum_i P(e_k | f_{i,k}) b_{k-1}(i)$$

For a 1-connector $p_m = 1$ since there is only one outcome. The cost of a connector $c_a$, $k(c)$, is a function of the action cost $c_a$ and the probabilities $p_m$ for each node $m$ in the set of successor nodes $suc(e)$.

$$k(c) = c_a + \sum_{m \in suc(e)} p_m q(m)$$

When the algorithm has expanded a node $n$, the cost $q_n$ is updated such that

$$q_n = \min_{c \in outg(n)} k(c)$$

where $outg(n)$ is the set of all outgoing connectors from $n$. The connector $c$ that minimizes (17) is included in a partial solution of the subproblem rooted in $n$. Until a partial solution is found for the root node, the same procedure is done for all predecessors of $n$ in $S$. When this is done, the partial solution for the root node is now the new best partial solution.

Recall that in the problem formulation in Section 2 we said that the terminal node in a troubleshooting strategy is successful only if the previous action is a successful function control. This means that even if we believe that the system is free of faults we still have to make the function control to complete the troubleshooting. Let this action be $a$ with the cost $c^{fc}$.
Then the cost $q_n$ of a node $n$ that is terminal in the explicit graph $G$ labeled with the goal state $S = (b, O)$ is

$$q_n = \begin{cases} 0 & \text{if } a \in O \\ c_i^f & \text{otherwise} \end{cases} \quad (18)$$

Note that if have the function control $a \in O$ in the goal state, this action must have been the most recently performed action.

A terminal node $n$ in $G'$ labeled with a state $S_n$ that is not a goal state cannot be a terminal node in $G$. The cost of this node is given by a heuristic function $h.$

$$q_n = h(S_n) \quad (19)$$

### 5.6 Searching with Limited Time

Finding an optimal solution to a large AND/OR graph cannot be done efficiently even with a fairly good heuristic. If we want the algorithm to come up with a solution while the user is waiting, it must finish in reasonable time. Therefore, after a certain time $T$ we will forbid the expansion of more nodes. Instead, the cost of the nodes, that should have been expanded, is set to a certain cut-off cost and the best partial solution $\tilde{s}$ is returned. This makes our search incomplete and the solution can no longer be guaranteed to be optimal. In [Sadikov and Bratko, 2006] it is argued that an optimistic heuristic function can degrade the result when used with incomplete search methods. Therefore, we propose to use a cut-off cost that gives us a pessimistic estimate of the optimal cost-to-go. The troubleshooting strategy based on efficiencies $s_{\text{eff}}$ described in Section 4.1 has this property. Since $s_{\text{eff}}$ assumes that the system is faulty, the cost of making the function control $c_i^f$ is added, if there is a non-zero probability that the system is free of faults. If the system has a fault, the expected cost is calculated using (4). For a node $n$ with the state $S_n = (b_n, O_n)$, let the integers $a_1, a_2, \ldots, a_m$ be the indexes of components ordered by efficiencies in descending order. Then $n$ is cut off the cost $q_n$ of that node is set to be

$$q_n = [b_n(0)]c_i^f + (1-b_n(0))\sum_{i=1}^{m}(b_n(a_i)(c_i^r+c_i^f+\sum_{j=1}^{i}c_j^r)) \quad (20)$$

where $c_j^r$ and $c_j^r$ are action costs of observing respectively repairing component $j$.

### 6 The Heuristic Function

The AO* algorithm will find the optimal solution provided an admissible heuristic function, i.e. a function that never overestimates the optimal cost-to-go from a node. As concluded in the previous section, when searching with limited time, the solution is no longer guaranteed to be optimal. Therefore, it is not necessary that the heuristic function is admissible. The aim of the new heuristic function is to minimize the relative error in the estimated cost-to-go.

In [Vomlelová and Vomlel, 2000] an admissible heuristic function is used together with AO* for solving the troubleshooting problem in the domain of home electronics. It is derived from a relaxation of the troubleshooting problem where we can make a "perfect" observing action that points out the true underlying component fault at no cost. The optimal cost of repair of the relaxed problem is easily calculated.

**A Heuristic for Near-Optimal Troubleshooting Using AO***

It is the cost of repairing the faulty component weighted with its probability plus the cost of a final function control. It is well known that admissible heuristics can be acquired by solving a relaxation of the problem optimally [Russell and Norvig, 2003]. Let $h_1 : S \rightarrow \mathbb{R}$ be this heuristic. For a system with $n$ components and the state $S = (b, O)$, i.e.

$$h_1(S) = c_i^f + \sum_{i=1}^{n}b(i)c_i^r \quad (21)$$

where $c_i^f$ is the cost of the function control and $c_i^r$ is the cost of repairing component $i$.

A problem with this heuristic is that it only considers the cost of the repair actions. This means that, if many observing actions are included in the optimal troubleshooting strategy, $h_1$ returns a value much too low. This is the case for the fuel injection system described in Section 3. Therefore we need a stronger heuristic.

A feature of the troubleshooting problem that is ignored by $h_1$ is our uncertainty of which component is faulty. The entropy of the probability distribution can be used as a measure of this uncertainty [Gray, 1990]. Let $S = (b, O)$ be the state of a node. Then the entropy of the probability distribution in $b, H(b)$, is given by

$$H(b) = -\sum_{i=0}^{n}b(i)\log b(i) \quad (22)$$

where $n$ is the number of components.

In experiments on the fuel injection system we have measured the optimal cost-to-go $q_n^*$ and the state $S_n = (b_n, O_n)$ of every node $n$ for which the optimal solution could be found using the $h_1$ heuristic. In Figure 4 we have plotted $q_n^*-h_1(S_n)$ against $H(b_n)$. As $H(b_n)$ grows a linear trend in the difference $q_n^*-h_1(S_n)$ is visible. This is caused by the extra observing actions needed to isolate component faults.

![Figure 4: Plot of $q_n^*-h_1(S_n)$ against $H(S_n)$ of 4250 measured nodes. Darker areas indicate a higher density of measurements.](image)

The heuristic function that we propose exploits this linear trend to get a better approximation of the estimated cost-to-go. The linear trend is modeled by a parameter $\hat{c}_H$ that describes the estimated cost of reducing entropy. We will call this the **entropy cost**. The new heuristic function $h_2 : S \rightarrow \mathbb{R}$ is defined as

$$h_2(S) = h_1(S) + \hat{c}_H H(S) \quad (23)$$
As with $h_1$, this heuristic does not require any expensive computation but it is not admissible and it requires a set of training data from the problem domain so that a value of $\hat{c}_H$ can be assigned.

We want to keep the relative error $\varepsilon(n)$ of the optimal cost-to-go $q^*_n$ and the heuristic value $h_2(S_n)$ minimal for every node $n$, i.e.

$$
\varepsilon(n) = \frac{|q^*_n - h_2(S_n)|}{q^*_n}
$$

(24)

We do this by fitting the parameter $\hat{c}_H$ in (23) to data from a training set of simpler problems in which $q^*_n$ is known using linear regression. For the fuel injection system in our case study this value was 60.3 and it is indicated by a dashed line in Figure 4.

7 Empirical Evaluation

To evaluate the new heuristic we will study how the relative error and the size of the search graph grows with problem size. We will also study how the new heuristic affects the performance of troubleshooting when we have limited time.

7.1 Relative Error

We can only measure the relative error on problems that can be solved optimally. These problem instances are obtained by making a series of random actions on the case study until the remaining problem is small enough to be solved using AO* with the admissible heuristic $h_1$. As a measurement of problem size we use the number of applicable actions available.

We will use as a reference a heuristic that is based on an analogy to the Huffman coding problem which is used with AO* to solve the test sequencing problem [Raghavan et al., 1999]. Let $c_1, c_2, \ldots$ be the costs of the observing actions that are applicable in the state $S$ ordered such that $c_1 \leq c_2 \leq \ldots$. Then the heuristic function $h_3$ is given by

$$
h_3(S) = \sum_{i=1}^{\lceil H(S) \rceil} c_i + (H(S) - \lceil H(S) \rceil)c_{\lceil H(S) \rceil + 1}
$$

(25)

Since the test sequencing problem does not involve repair actions we will combine $h_3$ and $h_2$ to get a more fair comparison. We will call the resulting heuristic $h_4$:

$$
h_4(S) = h_1(S) + h_3(S)
$$

(26)

Figure 5 shows a comparison of the relative errors of $h_1, h_2$, and $h_4$ on a different data set than the one used to calculate $\hat{c}_H$ with a problem size varying from 3 to 12.

Since the parameter $\hat{c}_H$ is designed to minimize the relative error, this value is the lowest for the $h_2$ heuristic. The relative error is approximately constant for all problem sizes in the experiment which indicates that $h_2$ is equally strong also for larger problem sizes.

7.2 Size of the Search Graph

To show the difference in the ability to focus the search, we have measured the number of created nodes $G'$ and the relative error from optimum $\delta$ when the algorithm is run with different heuristics on solvable problems with growing size. Let $ECR_i$ be the expected cost of the solution found using the heuristic $h_i$ and let $ECR^*$ be the expected cost of the optimal solution. Then the relative error of the heuristic $h_i$ is

$$
\delta(h_i) = \frac{ECR^* - ECR_i}{ECR^*}
$$

(27)

Since $h_1$ is admissible, $ECR_1 = ECR^*$. The sizes of the search graphs for $h_1, h_2$, and $h_4$ are shown in Figure 6. The relative error $\delta$ was at all times below 0.01 for both $h_2$ and $h_4$.

7.3 Troubleshooting with Limited Time

When troubleshooting with limited time we will only have a partial solution. The first action in this solution is performed on the system and a new partial solution is calculated. In this experiment, we will compare our version of AO* with limited time using $h_1$ and $h_2$ with the greedy two step look-ahead troubleshooting algorithm presented in [Langseth and Jensen, 2002].

We will let the troubleshooting algorithms troubleshoot the fuel injection system with predefined hidden faults. The system will respond to actions according to the probabilistic model. The troubleshooting stops when the correct component is repaired and a confirming functional control is made.
Each algorithm is allowed 10 seconds to decide the next action to perform. In 10 seconds the search algorithm expands approximately 30000 nodes. The mean cost of repair is measured for 100 randomly generated problem instances. The results are shown in Table 1.

<table>
<thead>
<tr>
<th>Time limited AO* using $h_1$</th>
<th>556.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time limited AO* using $h_2$</td>
<td>476.91</td>
</tr>
<tr>
<td>Time limited AO* using $h_4$</td>
<td>518.52</td>
</tr>
<tr>
<td>Greedy two step look-ahead</td>
<td>618.57</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the mean cost of repair for the troubleshooting algorithms.

The greedy two step search found its solutions within milliseconds, but since a function control always is required after each repair action the costs became higher. The time limited search algorithm is not constrained by this and thereby the costs were less. The $h_2$ measured the lowest costs. This is mainly due to that larger parts of the search graph could be explored during the 10 seconds.

8 Conclusions

We have shown how the AO* algorithm can be used to solve the troubleshooting problem on a case study from the automotive domain. We have shown that it can be used to recommend troubleshooting actions within a fixed time limit. We have presented a new entropy based heuristic. In the empirical evaluations we have shown that, by using this heuristic, the troubleshooting costs can be reduced compared to when using the greedy two-step look-ahead algorithm or any of the heuristics $h_1$ and $h_4$.

Acknowledgments

For fruitful discussions, we would like to thank our colleagues at the Dept. of Computer and Information Science at Linköping University, in particular Jonas Kvarnström and Per Nyblom. We would also like to thank our coworkers at Scania for all the support. This research has been funded by VINNOVA V-ICT and Scania CV AB.

References


Diagnosing Dependent Failures in the Context of Consistency-based Diagnosis

Jörg Weber and Franz Wotawa
Institute for Software Technology, Graz University of Technology
{jweber,wotawa}@ist.tugraz.at

Abstract

Consistency-based diagnosis approaches usually assume that components fail independently, i.e., that any abnormal behavior of a component is the consequence of an internal fault. Dependent failures occur when the behavior of a faulty component causes the failure of other components as well. We provide a general discussion on dependent failures and the shortcomings of common model-based diagnosis approaches in systems with dependent failures. We propose a model which makes the dependencies between components explicit, and we provide algorithms for computing hypotheses which indicate the causal order of the failures.

1 Introduction

Model-based diagnosis (MBD) approaches which follow the consistency-based diagnosis paradigm [Reiter, 1987; de Kleer and Williams, 1987; de Kleer et al., 1992] usually assume that components fail independently, i.e., that any abnormal behavior of a component is the consequence of an internal fault. Although some researchers have acknowledged that components may fail dependently (e.g., [de Kleer, 1990]), there are very few works which have addressed this issue.

With the term dependent failure we denote cascades of failures which happen when a component, the cause of the cascade (CoC), fails due to an internal fault and when this failure causes the failure of other components as well. It is normal that the failure of a component leads to the propagation of unexpected values/events, which are not predicted by nominal system behavior, throughout the system; however, in systems with dependent failures it may happen that some components suffer from persistent damage after unexpected occurrences at their inputs. In physical systems, phenomena like overvoltages, high pressure, heat, etc., may harm those components which have not been designed to sustain such contingencies.

In most existing approaches the independence assumption is reflected in at least two ways: first, the focusing criteria rely on it; second, the multiple-fault diagnoses do not indicate any dependencies between the failures. Many approaches compute the (subset)-minimal diagnoses or the minimal cardinality diagnoses. This is perfectly justified in many systems without dependent failures, as multiple independent failures are often very unlikely; however, as we will demonstrate below, in case of dependent failures those focusing criteria may miss failed components. Even though we cannot expect to find all component failures, we should at least seek to determine all possible causes of a cascade of failures (i.e., the CoC’s). Furthermore, the obtained results should state the dependencies between the failures, since this information is often essential for a successful recovery of the system.

Our main contribution is threefold. First, we provide a general discussion on dependent failures and their consequences (Section 2). We also investigate the notion of ”abnormality”, which may be controversial in this context. Second, we propose to enhance the system model with an additional model, the cascading failure graph (CFG), which explicitly captures possible dependencies between component failures (Section 3). We provide a formalization of the semantics of this model which also allows to ensure that the CFG is consistent with the system description (SD). Relying on the assumption that all multiple failures have a single cause, it is now possible to focus only on those minimal diagnoses which may be attributed to a single cause. Third, Section 4 describes an algorithm which, for each of the obtained minimal diagnoses, generates failure cascade hypotheses and evaluates which of those hypotheses are consistent with the observations. The main goal is to find the plausible causes for any minimal diagnosis and one or more consistent failure cascade hypotheses for each of those causes. We also provide a brief discussion on complexity issues (Section 5).

An important point is that the resulting hypotheses correspond to diagnoses which may be non-(subset)-minimal, i.e., they may consider components as failed which are not included in the minimal diagnoses, and they make the dependencies of the failures explicit.

Our proposal is built upon Reiter’s well-known diagnosis framework. We expect that our work can contribute to improve the diagnostic results of MBD approaches which are based on this framework when applied to systems in which dependent failures may occur. In particular we expect that this holds when repair is the ultimate purpose of the diagnosis system.

This research was partially funded by the Austrian Science Fund (FWF) under grant P20199-N15 and by the Australian Research Council under grant DP0560183.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
2 Discussion: Dependent Failures

Figure 1 depicts a circuit which will serve as a running example. Note that it is similar to examples in [Struss and Dressler, 1989; Friedrich et al., 1990a]. The voltage magnitudes \( u_{xy} \) are modelled in a qualitative way, i.e., we only distinguish between zero, norm, and high. The latter means "higher than desired"; in particular, \( u_{bi} = \text{high} \) indicates that the bulb \( B_i \) is exposed to a voltage which exceeds the range it was designed for (e.g., > 230 V). The system has two inputs \( s_{c1} \) and \( s_{c2} \), which are either on or off. They control the state of the switches. The filaments of the bulbs are either ok or broken. \( R \) is a series resistor which limits the voltage at the bulbs. As usual, the logical system description \( SD \) captures the nominal behavior of components using the predicate \( AB \) which denotes "abnormal":

\[
\begin{align*}
\neg AB(V) & \rightarrow (u_x = \text{norm}) \\
AB(R) \land (u_x = x) & \rightarrow (u_x = x), x \in \{ \text{zero, norm, high} \} \\
AB(R) & \rightarrow (u_x = \text{low}) \\
\neg AB(S_i) \land (s_{ci} = \text{on}) & \rightarrow (u_{bi} = u_x) \\
\neg AB(S_i) \land (s_{ci} = \text{off}) & \rightarrow (u_{bi} = \text{zero}) \\
\neg AB(B_i) & \rightarrow (fil_i = \text{ok}) \\
(fil_i = \text{ok}) \land (u_{bi} \neq \text{high}) & \rightarrow (\text{light}_i = \text{on}) \\
(u_i = \text{low}) \land (u_{bi} \neq \text{low}) & \rightarrow \bot \\
(u_i = \text{norm}) \land (u_{bi} = \text{high}) & \rightarrow \bot
\end{align*}
\]

We also assume that there are domain closure axioms \( (1) \) which restrict the possible values for each variable. If every component works correctly and both system inputs \( s_{c1} \) and \( s_{c2} \) are on, then all voltages in the model have the value \( \text{norm} \), and the two bulbs light. Moreover, note that a bulb \( B_i \) also lights when its input voltage \( u_{bi} \) is \( \text{high} \), as long as its filament is \( \text{ok} \). For now we presume that the resistor has only one possible faulty behavior: when it fails, then it exhibits an extremely high resistance. The last two sentences in \( (1) \) are physical impossibility axioms [Friedrich et al., 1990b] expressing that the switches and bulbs are not able to produce any additional voltage.

Now suppose that \( V \) fails in a way s.t. it produces a voltage significantly higher than expected, i.e., \( u_x = \text{high} \). Clearly, depending on the extent of this deviation, this will eventually destroy the bulbs, as the lifespan of a filament strongly decreases with higher voltage. Hence, a fault in \( V \) may be the cause of \( fil_i = \text{broken} \), and consequently it may be the cause of \( AB(B_i) \). In such a case, \( V \) is the CoC, the cause of the cascade of failures. The model in \( (1) \) does not reflect these facts.

The classical MBD approaches, which rely on the failure independence assumption, do not take into account that the abnormality of components (or more generally, fault modes of components) may be caused by external circumstances. This leads us to the important question whether a broken filament should always be regarded as abnormal behavior. Suppose we would modify the behavior model of the bulbs:

\[
\neg AB(B_i) \land (u_{bi} \neq \text{high}) \rightarrow (fil_i = \text{ok})
\]

This modification would mean that a bulb can be viewed as not abnormal even if its filament is broken, given that its input voltage is high; in other words, in this case a broken filament is a correct (expected) behavior of a bulb. If we adopted this view, then our notion of dependent failures would become meaningless – what we call a dependent failure would actually be a normal reaction of a component to an unexpected and damaging input.

In MBD a component is considered as abnormal when its actual behavior deviates from its specification; however, it seems that it is not always obvious what we should define as correct behavior. We argue that the definition of correct and abnormal behavior depends on the purpose of the diagnosis, as diagnosis always serves some motive like repair.

When repair is the purpose, then any component which needs to be repaired in order to restore the full system functionality should be regarded as abnormal. Consequently, the nominal behavior of a component should also state basic conditions (related solely to this component) whose violation indicates the persistent inability of the component to serve its desired purpose. Note that this discussion indicates that also the system and its expected functionality may have an impact on the behavior models of single components. Similar arguments are brought in [Struss et al., 2000].

Our work particularly aims at diagnosis applications whose purpose is repair. Hence, it is obvious that a broken filament should be considered as abnormal behavior of a bulb and that the model in \( (1) \) better serves our needs than \( (2) \).

The following definition is needed for the subsequent considerations. \( SD \) is the logical system description, \( COMP \) the set of components, and \( OBS \) a set of observations [Reiter, 1987]:

**Definition 1 (Diagnosis and Minimal Diagnosis)** A diagnosis for \( (SD, COMP, OBS) \) is a set \( \Delta \subseteq COMP \) s.t. \( SD \cup OBS \cup \{ AB(c) | c \in \Delta \} \) is consistent. \( \Delta \) is (subset-)minimal iff no proper subset of it is a diagnosis.

Let us consider the following scenario: \( OBS = \{ s_{c1} = \text{sc2 = on, light1 = light2 = off, fil1 = fil2 = broken} \} \). Then we obtain the minimal diagnosis \( \Delta = \{ B_1, B_2 \} \). Based on this result, one may attempt to restore the system by replacing both bulbs with new ones (note that collecting additional measurements is often expensive or even impossible). However, if the bulbs have failed dependently due to a fault in \( V \), then the new bulbs will soon fail again, as the actual cause remains faulty; i.e., \( V \) still produces a very high and damaging voltage.

This simple example shows that the focus on minimal diagnoses is questionable in systems with dependent failures. In systems without dependent failures, the minimal diagnosis \( \Delta = \{ B_1, B_2 \} \) should be clearly preferred to the non-minimal diagnosis \( \Delta' = \{ V, B_1, B_2 \} \), as multiple independent failures are unlikely, but this does not hold for systems
with dependent failures, where multiple failures can often be attributed to a single cause. In this scenario, an experienced human, who is aware of the possible failure dependencies, would come up with the hypothesis that \( V \) is the cause of the cascade (CoC), and maybe perform additional measurements to confirm or refute this hypothesis.

As we will see below, a failure cascade hypothesis states the assumed failure dependencies using a predicate \( DF \) (“dependent failure”). In this example, our approach would generate the hypothesis \( H_{V, \Delta} = \{ DF(V, B_1), DF(V, B_2) \} \), meaning that \( V \) is the CoC of the minimal diagnosis \( \Delta = \{ B_1, B_2 \} \) and that \( B_1 \) and \( B_2 \) have failed due to \( V \).

There are two crucial points. First, \( H_{V, \Delta} \) also considers \( V \) as abnormal, although it is not contained in \( \Delta \); i.e., \( H_{V, \Delta} \) corresponds to a non-minimal diagnosis \( \Delta' = \{ V, B_1, B_2 \} \), which is a superset of \( \Delta \). Second, \( H_{V, \Delta} \) also indicates the failure dependencies, i.e., the causal order of the cascade of failures. This is very important as in many systems this causal order also influences the order in which the components must be repaired; in particular, the CoC must often be repaired first. Here, \( V \) should be repaired before the bulbs are replaced, because otherwise there is the risk that a new bulb is destroyed again by the faulty voltage source. Moreover, the knowledge that a certain component has failed due to external circumstances rather than an internal fault may also be useful, e.g., for the assessment of the reliability of components.

Now we consider the case when we have full observability and \( OBS = \{ u_i = \text{high}, fil_1 = \text{fil} 2 = \text{broken}, \ldots \} \) which yields a single minimal diagnosis \( \Delta = \{ V, B_1, B_2 \} \). Although this result comprises all components which have actually failed, it still does not indicate the failure dependencies which we want to know. Finally, note that if we modified the model as shown in (2), then we would get the two minimal diagnoses \( \Delta_1 = \{ V \} \) and \( \Delta_2 = \{ B_1, B_2 \} \). Neither of these results is satisfying, as \( \Delta_1 \) does not contain the two bulbs which have obviously failed and should be repaired.

The discussions above show that if knowledge about possible failure dependencies exists, the diagnosis can be improved by an approach which takes those dependencies into account and which is also able to provide results which state the causal order of failures. Moreover, it should be possible to logically refute those failure cascade hypotheses which are inconsistent with the observations.

### 3 Failure Cascade Hypotheses

We propose to explicitly model the possible failure dependencies in a cascading failure graph (CFG), a model which is separated from the system description \( SD \). This concept is similar to the hidden interaction models for the diagnosis of structural faults [Böttcher, 1995], which explicitly describes unintended interactions. One advantage of this separation is that the complexity of computing the minimal diagnoses remains the same, which is an important point, e.g., if ones follows the strategy to consider dependent failures only when no single-fault diagnosis is found.

**Definition 2 (System)** A system is a tuple \( (SD, CFG, COMP) \). As usual, \( SD \) contains the behavior models of components relying on the \( AB \) predicate. We also allow the specification of faulty behavior [de Kleer et al., 1992].

Figure 2(a) depicts a very abstract CFG for the circuit. The CFG is a causal model whose edges represent MAY relationships, similar to those in [Console et al., 1989]. Intuitively, the model in this figure indicates that \( AB(V) \) may cause \( AB(B_1) \) and/or \( AB(B_2) \). As discussed in [Console et al., 1989], the usage of MAY relationships denotes some incompleteness of the model: we know that \( AB(V) \) can, under certain conditions, lead to \( AB(B_i) \), but either not all of those conditions are known or we consider it inappropriate to model at such a detailed level: abstract models allow for more efficient reasoning, and they reduce the modelling effort.

**Definition 3 (Cascading Failure Graph (CFG))** A cascading failure graph (CFG) is a directed acyclic graph (DAG) whose nodes are conjunctions of literals$^1$. Each node contains at most one \( AB \) literal which must be positive. For every component \( c \), the literal \( AB(c) \) may occur at most once in the CFG. Moreover, each edge is labelled with an abstract condition symbol \( \alpha \) [Console et al., 1989].

The abstracted condition symbols abstract from the actual conditions which may be very complex or even unknown. An edge \( S_i \rightarrow S_j \) from node \( S_i \) to \( S_j \) corresponds to the logical sentence \( S_i \land \alpha \rightarrow S_j \). The semantics of a CFG is formally defined below.

The CFG in Fig. 2(a) indicates that the failure of one or both bulbs may be caused by \( AB(V) \). However, it does not contain further conditions which could be compared with the observations. Hence, the model can be refined as shown in Fig. 2(b): \( u_i = \text{high} \) is a possible consequence of \( AB(V) \), and it may lead to \( u_{ab} = \text{high} \) which may further lead to \( AB(B_i) \). The CFG is a partial description of what may happen in the course of cascades of failures. The edge \( \alpha_1 \) could be regarded as a fault model of \( V \), whereas \( \alpha_2 \) and \( \alpha_3 \) abstract from the local behavior of the components between \( V \) and \( B_i \). \( \alpha_4 \) and \( \alpha_5 \) are particularly remarkable, as they embody all conditions which must be fulfilled so that \( u_{ab} = \text{high} \) destroys the bulb \( B_i \); they abstract from the exact magnitude of the voltage, the period of time since \( u_{ab} \) has become \( \text{high} \), the age of the bulb, etc.

Also note that the literal \((fil_i = \text{broken})\) in the nodes \( AB(B_i) \land (fil_i = \text{broken}) \) is redundant, as \( AB(B_i) \) implies

---

$^1$We do not distinguish between the node itself and the associated conjunction of literals.
(fil = broken) in the system description, but it shows that it is possible to state additional conditions which must hold when a component fails dependently.

A component \( c_{i,1} \) may directly cause the dependent failure of a component \( c_{i,k} \) iff there is a direct dependency path from \( c_{i,1} \) to \( c_{i,k} \).

**Definition 4 (Dependency Path)** A path \( S_{i,1} \xrightarrow{\alpha_{i,1}} S_{i,2} \xrightarrow{\alpha_{i,2}} \ldots \xrightarrow{\alpha_{i,k-1}} S_{i,k} \) in CFG is a dependency path from \( c_{i,1} \) to \( c_{i,k} \) (\( k > 1 \)) iff \( S_{i,1} \) contains \( AB(c_{i,1}) \) and \( S_{i,k} \) contains \( AB(c_{i,k}) \). Moreover, if no AB literal occurs in any node \( S_{i,j} \) with \( 1 < j < k \), then it is a direct dependency path.

In the following we assume, for simplicity, that there is at most one direct dependency path between two components (the generalization to an arbitrary number is straightforward).

In Fig. 2 there are two direct dependency paths: from \( V \) to \( B_1 \) and from \( V \) to \( B_2 \).

**Definition 5 (Dependency Assumption)** For every pair \((c_{i,1}, c_{i,k})\) of components with a direct dependency path from \( c_{i,1} \) to \( c_{i,k} \) there is a dependency assumption \( DF(c_{i,1}, c_{i,k}) \), denoting that the failure of \( c_{i,1} \) has directly led to the failure of \( c_{i,k} \). The set of all dependency assumptions of a system is denoted by \( \Phi \).

In our example we have: \( \Phi = \{DF(V, B_1), DF(V, B_2)\} \).

The cascading failure model (CFM) captures the semantics of a CFG. It is automatically generated from the CFG:

**Definition 6 (Cascading Failure Model (CFM))** The cascading failure model (CFM) is a set of logical sentences. It is created as follows. For each edge \( S_i \rightarrow S_j \): add \( S_i \land \alpha \rightarrow S_j \) to CFM. Moreover, for every pair of components \((c_{i,1}, c_{i,k})\) with a direct dependency path \( c_{i,1} \rightarrow \ldots \rightarrow c_{i,k} \) from \( c_{i,1} \) to \( c_{i,k} \) add \( DF(c_{i,1}, c_{i,k}) \rightarrow S_{i,1} \land \alpha_{i,1} \land \alpha_{i,2} \land \ldots \land \alpha_{i,k-1} \) to CFM.

**Theorem 1** For every pair \((c_{i,1}, c_{i,k})\) with a direct dependency path \( c_{i,1} \rightarrow \ldots \rightarrow c_{i,k} \) from \( c_{i,1} \) to \( c_{i,k} \) the following holds:

\[
\text{CFM} \cup \{DF(c_{i,1}, c_{i,k})\} \models S_{i,1} \land \ldots \land S_{i,k}
\]

Proof. We proof that \( \text{CFM} \cup \{DF(c_{i,1}, c_{i,k})\} \models S_{i,1} \land \ldots \land S_{i,j} \) with \( 2 \leq j \leq k \) by induction over \( j \).

Base: \( j = 2 \): \( DF(c_{i,1}, c_{i,k}) \rightarrow S_{i,1} \land \alpha_{i,1} \land \ldots \), and CFM also contains \( S_{i,1} \land \alpha_{i,1} \rightarrow S_{i,2} \) [see Def. 6].

Induction: we assume that \( \text{CFM} \cup \{DF(c_{i,1}, c_{i,k})\} \models S_{i,1} \land \ldots \land S_{i,j} \) holds for \( j < k \). As CFM contains \( S_{i,j} \land \alpha_{i,j} \rightarrow S_{i,j+1} \), it is clear that \( S_{i,j+1} \) is also entailed by CFM with \( \{DF(c_{i,1}, c_{i,k})\} \).

From Theorem 1 it follows that

\[
\text{CFM} \cup \{DF(c_{i,1}, c_{i,k})\} \models AB(c_{i,1}) \land AB(c_{i,k})
\]

In our example, CFM contains the following sentences:

\[
AB(V) \land \alpha_1 \rightarrow (u_0 = \text{high}) \\
(u_0 = \text{low}) \land \alpha_2 \rightarrow (u_{i,2} = \text{high}) \\
\ldots \\
DF(V, B_1) \rightarrow AB(V) \land \alpha_1 \land \alpha_2 \land \alpha_4 \quad (3)
\]

In practice the CFG will often be incomplete; i.e., it may not contain all possible failure dependencies, and the descriptions of what happens in the course of a failure cascade will usually be incomplete, too. However, we demand that the CFG must be consistent with SD. Intuitively, this means that the assumption that a certain component \( c_{i,k} \) fails due to \( c_{i,1} \) must be consistent with the system description:

**Definition 7 (Validity of a CFG)** A CFG is valid iff the following holds for every assumption \( DF(c_{i,1}, c_{i,k}) \in \Phi \):

\[
SD \cup CFM \cup \{DF(c_{i,1}, c_{i,k})\} \not\models \bot
\]

This implies that every MAY relationship \( S_i \rightarrow S_j \) on any direct dependency path is basically feasible, i.e.: \( SD \cup \{S_i \land S_j\} \not\models \bot \).

E.g., suppose there would be an edge \( (u_v = \text{low}) \rightarrow (u_{i,2} = \text{high}) \). Then the CFG would be invalid, as \( (u_v = \text{low}) \land (u_{i,2} = \text{high}) \) is, due to the model for \( AB(R) \) and the impossibility axioms, inconsistent with the model in (1).

**Definition 8 (Failure Cascade Hypothesis)** A failure cascade hypothesis \( H \in \Phi \) is a set of dependency assumptions s.t. the following holds: if \( DF(c', c) \in H \), then there is no component \( c'' \) with \( DF(c'', c) \in H \).

This definition implies that the dependent failure of \( c \) can be directly caused only by a single component \( c' \). This is a simplification which we consider appropriate for most practical purposes.

**Definition 9 (Cause of a Cascade (CoC))** Given a hypothesis \( H \), a component \( c \) is a cause of a cascade (CoC) in \( H \) iff there is at least one component \( c' \) s.t. \( DF(c', c) \in H \) and there is no component \( c'' \) with \( DF(c'', c) \in H \).

E.g., \( H = \{DF(V, B_1), DF(V, B_2)\} \) there is only one CoC, namely \( V \). In general, a hypothesis may have multiple CoCs.

We introduce the notation \( \Gamma(H) \) to denote the set \( \Gamma(H) = \{c \mid DF(c', c) \in H \text{ or } DF(c', c) \in H\} \).

In other words, \( \Gamma(H) \) comprises exactly those components \( c \) for which \( CFM \cup H \models AB(c) \).

E.g., for \( H = \{DF(V, B_1), DF(V, B_2)\} \) we obtain \( \Gamma(H) = \{V, B_1, B_2\} \).

**Definition 10 (Consistency of a Hypothesis)** A hypothesis \( H \) is consistent iff

\[
SD \cup CFM \cup OBS \cup H \cup \{\neg AB(c) \mid c \in \Gamma\} \not\models \bot
\]

with \( \Gamma = COMP \setminus \Gamma(H) \).

**Theorem 2** If a hypothesis \( H \) is consistent, then \( \Gamma(H) \) is a diagnosis.

Proof. From Def. 10 it follows that \( SD \cup OBS \cup \{AB(c) \mid c \in \Gamma(H)\} \cup \{\neg AB(c) \mid c \in COMP \setminus \Gamma(H)\} \) is consistent. Hence, according to Def. 1, \( \Gamma(H) \) is a diagnosis.

**Corollary 3** A necessary (but not sufficient) condition for a hypothesis \( H \) to be consistent is that \( \Gamma(H) \) is an (improper) superset of a minimal diagnosis \( \Delta \).

[^2]: For brevity we will often simply write "hypothesis".

Proof. Suppose that $\Gamma(H)$ is not an improper superset of a minimal diagnosis. According to [Reiter, 1987; de Kleer et al., 1992], it follows that $\Gamma(H)$ is not a diagnosis. Now we can conclude from Theorem 2 that the hypothesis $H$ cannot be consistent.

The main question is which hypotheses should be generated and tested for consistency by a diagnosis system. We propose to focus on hypotheses which have a single CoC; this corresponds to the assumption that all multiple failures have a single cause. This is similar to the focus on minimal cardinality diagnoses (e.g., [de Kleer, 1990]): in both cases, multiple independent failures are considered very unlikely. Moreover, the author of [Böttcher, 1995] also proposed to prefer those multiple-fault diagnoses which can be attributed to a single structural fault. The focus on hypotheses with a single CoC motivates the following definition:

**Definition 11** (σ-hypothesis) A σ-hypothesis $H_{c,\Delta}$ which relates to a component $c$ and a (non-empty) minimal diagnosis $\Delta$, is a failure cascade hypothesis which has exactly one CoC, namely $\gamma$. Moreover, the empty set is not a σ-hypothesis.

It is important to note that a σ-hypothesis $H_{c,\Delta}$ may consider components as failed which are not included in $\Delta$. Moreover, the empty set is not a σ-hypothesis. E.g., given the CFG’s in Fig. 2, the diagnosis $\Delta = \{B_1, B_2\}$ has exactly one σ-hypothesis $H_{V,\Delta} = \{DF(V, B_1), DF(V, B_2)\}$.

We propose to compute only those minimal diagnoses which may have a single cause, to generate σ-hypotheses for these diagnoses, and to check the consistency of the hypotheses. In general, a specific minimal diagnosis $\Delta$ may have several possible causes, and for each potential cause there may be a large number of σ-hypotheses. In many applications it will not be possible to check the consistency of all σ-hypotheses, and so we propose the strategy to seek (at least) one consistent σ-hypothesis for each possible cause of a minimal diagnosis. The reason behind this strategy is the observation that finding and repairing the ultimate cause of a cascade of failures is, in many domains, crucial for the successful repair of the independently failed components, as we have already seen in the circuit example.

If there is no consistent σ-hypothesis for a possible CoC $c$ and a minimal diagnosis $\Delta$, then the cause $c$ can be regarded as refuted. Moreover, single-fault diagnoses or multiple-fault diagnoses which have consistent σ-hypotheses are preferred over multiple-fault diagnoses which cannot be attributed to a single cause.

### 4 Finding Consistent σ-hypotheses

For illustration purposes we make, for the rest of this paper, the assumption that the component $R$ may also fail in a way s.t. its resistance becomes very low (e.g., this is the case when $R$ actually represents the internal resistance of a more complex device which may have a short). I.e., we remove the sentence $AB(R) \rightarrow (u_s = low)$ from $SD$ and add an impossibility axiom instead: $(u_s = low) \land (u_s \neq low) \rightarrow \bot$. Figure 3 depicts a CFG for our example circuit which now considers all components. It also models that $R$ may fail in dependence of $V$, and that a failure of $R$ may lead to an overvoltage at the switches and the bulbs. Furthermore, we assume that an overvoltage at the switches may also cause a failure of $S_1$ and/or $S_2$.

![Figure 3: An extensive CFG for the circuit. The edge labels are omitted.](image)

We obtain 9 dependency assumptions: $\Phi = \{DF(V, R), DF(V, S_1), DF(V, S_2), DF(V, B_1), DF(V, B_2), DF(R, B_1), DF(R, B_2)\}$. The resulting example relates to the scenario $OBS = \{f_{d1} = f_{d2} = broken, light_1 = light_2 = of\}$, $sce_1 = on, sce_2 = off\}$.

The first step is to compute those minimal diagnoses which may have a single cause. For $COMP' \subseteq COMP$ we define the notation:

$$\gamma(\text{COMP}') = \{c' \mid \text{for all } c' \in \text{COMP}', \text{ either } c' = c \text{ or there is a dependency path from } c \text{ to } c'\}$$

(4)

Note that the dependency path may be direct or indirect. If the diagnosis $\Delta$ can be attributed to single causes, then $\gamma(\Delta)$ contains all possible CoC’s of the σ-hypotheses of $\Delta$; otherwise, $\gamma(\Delta)$ is empty. E.g., $\gamma(\{B_1\}) = \{V, R, B_1\}$, $\gamma(\{B_1, B_2\}) = \{V, R\}$, $\gamma(\{V, R, B_1\}) = \{V\}$, $\gamma(\{V\}) = \{V\}$. It can be shown that the following holds:

$$\gamma(\text{COMP}') = \bigcap_{c' \in \text{COMP}'} \gamma(\{c'\})$$

(5)

If $D$ denotes the set of all minimal diagnoses, we only want to compute the set $D_1 = \{\Delta \mid \Delta \in D \text{ and } \gamma(\Delta) \neq \emptyset\}$. In our scenario we obtain $D = \{\Delta\} \text{ with } \Delta = \{B_1, B_2\}$. Moreover, $\gamma(\Delta) = \{V, R\}$, and hence $D_1 = D$. However, in general $D_1 \subseteq D$ holds. We sketch how Reiter’s Hitting Set algorithm [Reiter, 1987] (and other algorithms following similar ideas) can be modified to compute only $D_1$ instead of $D$:

1. Precompile $\gamma(\{c\})$ for every $c \in \text{COMP}$. The size of the resulting data structure is $O(|\text{COMP}|^2)$.
2. Each time a diagnosis candidate $\Delta'$ is logically refuted (i.e., the consistency check as indicated in Def. 1 leads to a contradiction) and a new candidate with $\Delta'' = \Delta' \cup \{c\}$ is created: $\gamma(\Delta'') = \gamma(\Delta') \cap \gamma(\{c\})$.
3. If $\gamma(\Delta'') = \emptyset$, then discard the candidate $\Delta''$ and do not create any candidates which are supersets of $\Delta''$.

The generation of σ-hypotheses relies on the notion of DF-path.

**Definition 12** (DF-path) A DF-path $\psi$ from a component $c_{i,1}$ to $c_{i,k}$ ($k > 1$) is a set $\psi = \{DF(c_{i,1}, c_{i,2}), DF(c_{i,2}, c_{i,3}), \ldots, DF(c_{i,k-1}, c_{i,k})\}$.

E.g., the set $\{DF(V, R), DF(R, B_1)\}$ is a DF-path. Obviously, a DF-path is also a hypothesis. Moreover, we observe that a DF-path from a component $c_{i,1}$ to $c_{i,k}$ can exist only if there is a dependency path in the CFG from $c_{i,1}$ to $c_{i,k}$. 

Algorithm 1: Compute all core hypotheses related to a CoC \( c \) and a minimal diagnosis \( \Delta \)

**Input:** \( c \) and \( \Delta \) (note that \( c \in \gamma(\Delta) \))

**Output:** A set \( HYP_{c,\Delta} \) comprising all core hypotheses

1. (1) For all \( c' \in \Delta \setminus \{c\} \):
   \[
   \psi_{c,c'} := \{ \psi_{c,c'}' | \psi_{c,c'}' \text{ is a DF-path from } c \text{ to } c' \}
   \]
2. Generate the set \( HYP_{c,\Delta} := \)
   \[
   \left\{ H_{c,\Delta} | H_{c,\Delta} = \bigcup_{c' \in \Delta \setminus \{c\}} \psi_{c,c'} \cdot \psi_{c,c'}' \in \psi_{c,c'} \right\}
   \]
3. For all \( H_{c,\Delta} \in HYP_{c,\Delta} \): if there are components \( \tilde{c}, c', c'' \) (with \( c' \neq c'' \)) s.t. \( DF(c', \tilde{c}) \in H_{c,\Delta} \) and \( DF(c'', \tilde{c}) \in H_{c,\Delta} \): remove \( H_{c,\Delta} \) from \( HYP_{c,\Delta} \).

The following theorem states an important property of \( \sigma \)-hypotheses:

**Theorem 4** Any possible \( \sigma \)-hypothesis \( H_{c,\Delta} \) contains a DF-path \( \psi_{c,c'} \) from \( c \) to each \( c' \in \Delta \) with \( c' \neq c \) (i.e., \( \psi_{c,c'} \subseteq H_{c,\Delta} \)).

Proof. From Def. 11 it follows that \( DF(c', c') \in H_{c,\Delta} \) and/or \( DF(c', c') \in H_{c,\Delta} \) for each \( c' \in \Delta \) with \( c' \neq c \). In both cases, suppose there would be no DF-path from \( c \) to \( c' \).

If the first case, this would mean that \( c' \neq c \) and that there is no DF-path from \( c \) to \( c' \). It follows that \( c'' \neq c \) is a CoC in \( H_{c,\Delta} \), or that there is another component \( c'' \neq c \) which has a DF-path to \( c'' \) in \( H_{c,\Delta} \) and which is a CoC. However, then there would be more than one CoC which contradicts Def. 11.

Analogous considerations apply to the second case. \( \square \)

In order to determine all possible single causes of the failures, we seek to find, for every \( \Delta \in D_{\sigma} \), (at least) one consistent \( \sigma \)-hypothesis for each \( c \in \gamma(\Delta) \). We consider the core hypotheses first:

**Definition 13 (Core Hypothesis)** A \( \sigma \)-hypothesis \( H_{c,\Delta} \) is a core hypothesis for \( c \) and \( \Delta \) iff it is subset-minimal, i.e., no proper subset of it is a \( \sigma \)-hypothesis related to \( c \) and \( \Delta \).

E.g., for \( \Delta = \{B_1, B_2\} \) and the possible cause \( V \) there are 4 core hypotheses: \( HYP_{V,\Delta} = \{ H_{V,\Delta}, H_{V,\Delta}, H_{V,\Delta}, H_{V,\Delta} \} \) with
\[
H_{V,\Delta} = \{DF(V, B_1), DF(V, B_2)\},
H_{V,\Delta} = \{DF(V, R), DF(R, B_1), DF(R, B_2)\},
H_{V,\Delta} = \{DF(V, R), DF(R, B_1), DF(R, B_2)\},
H_{V,\Delta} = \{DF(V, R), DF(V, B_2), DF(R, B_1)\}.
\]

Algorithm 1 computes all core hypotheses related to a CoC \( c \) and a minimal diagnosis \( \Delta \). The resulting set \( HYP_{c,\Delta} \) comprises all possible unions of DF-paths from \( c \) to each \( c' \in \Delta \setminus \{c\} \) s.t. the resulting set is a hypothesis. Note that line (3) removes those sets of dependency assumptions which are not a hypothesis.

**Theorem 5** The set \( HYP_{c,\Delta} \) which is computed by Alg. 1 consists of all core hypotheses related to the CoC \( c \) and \( \Delta \).

We omit the proof due to space reasons.

In the above example, all of the 4 core hypotheses in \( HYP_{V,\Delta} \) are inconsistent: intuitively, all of them contain either \( DF(V, B_2) \) or \( DF(R, B_2) \) and thus predict \( u_{b_2} = high \) (see the CFG in Fig. 3), but they consider \( S_2 \) as not abnormal. Together with the observation \( s_{c_2} = o f f \) this contradicts the model \( \neg AB(S_2) \land (s_{c_2} = o f f) \rightarrow (u_{b_2} = zero) \).

Hence, it may also be necessary to generate and test supersets of core hypotheses; i.e., further components must be considered as failed. This can be done in a conflict-directed way:

**Definition 14 (Conflict Set)** A set of assumptions \( F \subseteq \{AB(c) | c \in COMP\} \cup \{\neg AB(c) | c \in COMP\} \) is a conflict set iff the following holds:
\[
SD \cup CFM \cup OBS \cup F \models \bot
\]

Similar to [Reiter, 1987], we assume that the reasoner which performs the consistency checks of hypotheses (see Def. 10) is able to return a conflict set \( F \) when the tested hypothesis is inconsistent. Using the obtained conflict set, a refuted hypothesis \( H_{c,\Delta} \) can be expanded; i.e., a set of successor \( \sigma \)-hypotheses is generated s.t. each of them attempts to resolve the conflict by adding an additional DF-path from \( c \) to a component which might have failed as well. Algorithm 2 generates all successor \( \sigma \)-hypotheses. Note that line (10) in Alg. 2 is similar to line (3) in Alg. 1: it removes those sets of dependency assumptions which are not a hypothesis.

In our example, the consistency check for, e.g., \( H_{V,\Delta}^{11} = \{DF(V, B_1), DF(V, B_2)\} \) would return \( F = \{DF(V, B_2), \neg AB(S_2)\} \). By expanding \( H_{V,\Delta} \) we obtain 2 successor hypotheses: \( H_{V,\Delta}^{11} = \{DF(V, S_2), DF(V, B_1), DF(V, B_2)\} \) and \( H_{V,\Delta}^{12} = \{DF(V, R), DF(R, S_2), DF(V, B_1), DF(V, B_2)\} \). Both of them are consistent.

The algorithm for finding a consistent \( \sigma \)-hypothesis (if one exists) related to a minimal diagnosis \( \Delta \) with a cause \( c \) is outlined in Alg. 3. Note that this algorithm may apply a preference criterion for selecting hypotheses candidates; e.g., a probabilistic criterion. Moreover, the algorithms can be improved s.t. the hypotheses are generated incrementally, hoping that a consistent \( \sigma \)-hypothesis for \( (c, \Delta) \) is quickly found, i.e., it is not necessary to create all \( \sigma \)-hypothesis for \( (c, \Delta) \).

Moreover, note that two different diagnoses \( \Delta_1 \) and \( \Delta_2 \) may share the same \( \sigma \)-hypotheses with a specific CoC \( c \) (i.e.,
Algorithm 3: Search for a consistent $\sigma$-hypothesis $H_{c,\Delta}$

**Input:** $c$ and $\Delta$ ($c \in \gamma(\Delta)$)

**Output:** a consistent $\sigma$-hypothesis $H_{c,\Delta}$ (only if one exists)

1. Execute Alg. 1, let the set $HY_{P_{c,\Delta}}$ contain the resulting core hypotheses
2. Create a queue $Q$, add every $H_{c,\Delta} \in HY_{P_{c,\Delta}}$ to $Q$
3. repeat until $Q$ is empty:
   (a) Fetch any $H_{c,\Delta}$ from $Q$ (note that there could be a preference criterion for selecting a hypothesis)
   (b) Check consistency of $H_{c,\Delta}$ (see Def. 10)
   (c) If it is consistent: return $H_{c,\Delta}$ (algorithm finished)
   (d) Else: call Alg. 2 for $H_{c,\Delta}$ and the obtained conflict set $F_{c}$; let $HY'_{P_{c,\Delta}}$ contain the results
   (e) For every $H'_{c,\Delta} \in HY'_{P_{c,\Delta}}$: check if $H'_{c,\Delta}$ is already in $Q$; if not: add $H'_{c,\Delta}$ to $Q$

$HY_{P_{c,\Delta}} \cap HY'_{P_{c,\Delta}} \neq \{\}$. It follows that the results obtained for one diagnosis may be re-used for the computation of $\sigma$-hypotheses for other diagnoses as well. We do not discuss this in more detail due to space reasons.

5 Complexity Issues

Suppose $l$ an upper bound for the length of DF-paths in the system, i.e., $|\Gamma(\psi)| \leq l$ for all possible DF-paths $\psi$. Moreover, let $b$ denote an upper bound for the number of components which may directly cause a failure of a component $c$, i.e.: $b = \max_{c \in \text{COMP}} |\text{deg}(c)|$ with $\text{deg}(c) = \{c' | DF(c',c) \in \Phi\}$. Finally, suppose that $|\Delta| \leq m$ for every minimal diagnosis $\Delta$. Then the following holds:

**Theorem 6** Given a minimal diagnosis $\Delta$ and $c \in \gamma(\Delta)$, the set $HY_{P_{c,\Delta}}$ of all core hypotheses related to the possible cause $c$ (the CoC) and $\Delta$ has a worst-case size of $O(b^{l+m})$.

**Proof.** Between $c$ and any $c' \in \Delta$, there are $O(b^{l+2})$ possible DF-paths with a length $\leq l$. As a core hypothesis for $c$ and $\Delta$ is the union of DF-paths from $c$ to every $c' \in \Delta \setminus \{c\}$, the number of possible core hypotheses related to $c$ and $\Delta$ is $O(b^{l+2+m}) = O(b^{l+m})$.

Even though this is a rough estimation, it indicates that the number of possible core hypotheses may be huge in the worst case, and the number of possible $\sigma$-hypotheses may be even higher. Our approach may be intractable if $l$ and $m$ are large, i.e., if the system is large and the cascades of failures propagate through large parts of the system. However, even in systems with many components our proposal may be applicable provided that the possible failure propagations are locally limited and hence $l \ll n$ and $m \ll n$. Also note that in real systems it is often the case that $b$ is very small (e.g., $b = 2$), or that there are many components which can not fail in dependence of other components.

Furthermore, as we propose to seek only one consistent $\sigma$-hypothesis for each minimal diagnosis $\Delta$ and a possible single cause $c$, it is very often not necessary to generate and test all possible $\sigma$-hypotheses (we have already mentioned the possibility to generate the hypotheses incrementally). Finally, the number of hypotheses to be created can be further reduced by applying heuristics; e.g., a troubleshooter could decide to consider only causes within a certain distance (in the system structure) to the components in the minimal diagnosis.

6 Related Research and Conclusion

In systems with dependent failures one could, rather than focusing on minimal or minimal cardinality diagnoses, use behavioral modes and focus on the most probable diagnoses as described in [de Kleer and Williams, 1989; de Kleer, 1991]. Even though those works assume that components fail independently, it would be theoretically possible to extend them by representing the failure dependencies between components as conditional probability distributions. However, this approach would have several drawbacks:

First, in most systems the exact dependent failure probabilities are not available. Although it is a common way to use rough estimations, we expect that finding appropriate values which yield the desired diagnostic results is much more difficult when there are dependencies between components. Moreover, Williams and Ragno [Williams and Ragno, 2007] present a search algorithm which is supposed to generate the most probable candidates in an efficient way, but this work heavily relies on the independence assumption. In general, if $n$ is the number of components and $k$ the number of modes per component, then there are $O(k^n)$ possible mode assignments. Third, there is still the issue that the obtained results should indicate the causal order of the failures (or at least the cause of the cascade).

There is little work in the context of consistency-based diagnosis which explicitly addresses dependent failures. The author of [Lucas, 2001] proposes a method for reasoning with uncertainty in MBD which enables stochastic dependencies among components; however, this work does not provide an algorithm for computing the most probable candidates, and dependent failures are not considered at the logical level. The authors of [Roos and Witteveen, 2005] discuss the dependency between the health state of an agent and the (ab-)normality of its actions; however, their notion of dependent failures is quite different from ours, and their work is tailored to the diagnosis of agent plans.

[Weber and Wotawa, 2007] is an earlier work which explicitly addresses the issue of dependent failures. Similar to our new proposal, it employs a graph, the Failure Dependency Graph (FDG), stating possible failure dependencies between components; i.e., every node in a FDG is a component. The earlier work lacks a deeper discussion on dependent failures, and the semantics of the FDG is unclear. By contrast, we provide a logical semantics for our new dependency model, and the formalization relies on the resulting logical model, the CFM. We also provide a formal criterion for the consistency of the CFG with the system description $SD$ (see Def. 7). Another shortcoming of the approach in [Weber and Wotawa, 2007] is that, in most cases, it computes much more results than our new approach does, hence it is computationally significantly more complex.

The issue of dependent failures is similar to faults in the structure of the system which lead to hidden interactions, e.g., the bridge fault [Davis, 1984; Preist and Welham, 1990; Böttcher, 1995; de Kleer, 2007]. 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.

We show that the focus on minimal diagnoses is not appropriate for systems with dependent failures. This issue is linked to the notion of abnormality which, as discussed above, is somehow controversial in this context. Moreover, we also show that the repair in such systems often requires knowledge about the failure dependencies; in particular, it is important to know which component fault has caused the cascade of failures. We propose a model which explicitly captures possible failure dependencies. It allows one to determine, for each minimal diagnosis, the possible causes of the cascade of failures, to generate corresponding failure cascade hypotheses, and to check their consistency. We propose to compute only those minimal diagnoses which may have a single cause. The resulting hypotheses may correspond to non-minimal diagnoses, and they state the causal order of failures.

In many cases a troubleshooter will consider dependent failures only if there are no single-fault diagnoses. Our approach accommodates this strategy, because the complexity of computing minimal diagnoses remains the same.

A limitation of our current approach is the fact that it does so far not allow for expressing cyclic dependencies because the CFG is an acyclic graph. However, we expect that our framework can be extended to directed graphs containing cycles: it is necessary to enhance the hypothesis generator with cycle detection capabilities s.t. the created hypotheses are cycle-free. Note that many existing model-based diagnosis approaches presume an acyclic system structure.

One could argue that a disadvantage of our approach is that our dependency model is device-specific and non-local. On the other hand, the same also holds in general for the widely known and approved concept of physical impossibility axioms [Friedrich et al., 1990b]. Anyway, future research should seek to generate the dependency models from local behavior models. This could only be achieved if very detailed behavior models exist which completely capture the propagation of failure symptoms throughout the system. Note that such detailed models are often not available, and they are often also inappropriate for practical purposes as they are too complex.

References


[de Kleer and Williams, 1989] Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In Proceed-

ings of the 11th International Joint Conf. on Artificial Intelligence, pages 1324–1330, 1989.


Posters
Machine Learning and Model Based Diagnosis using Possible Conflicts and System Decomposition.*

C. J. Alonso González †, J.J. Rodríguez ‡, O. Prieto †, B. Pulido †

† Depto. de Informática (GSI), Universidad de Valladolid, Spain. {calonso,oscapri,belar}@infor.uva.es
‡ Lenguajes y Sistemas Informáticos, Universidad de Burgos, Spain. jjrodriguez@ubu.es

Abstract

This work presents an on-line diagnosis algorithm for dynamic systems that combines model based diagnosis and machine learning techniques. The Possible Conflicts method is used to perform consistency based diagnosis. Possible conflicts are in charge of fault detection and isolation. Machine learning methods are used to induce time series classifiers, that are applied on line for fault identification. The main contribution of this work is that Possible Conflicts are used to decompose the physical system. This decomposition allows defining the input-output structure of an ensemble of classifiers. Hence the structural knowledge provided by the Possible Conflicts is exploited by the machine learning methods. Possible Conflict decomposition may be used for class selection or for class and attribute selection. Experimental results on a simulated pilot plant show that class selection has an important potential to increase the classifier accuracy for several learning algorithms. The effect of an additional attribute selection depends on the kind of machine learning method, although it improves the accuracy of the most precise classifiers, especially at the first stages of the diagnosis processes, just after a fault has been detected.

Keywords: System Decomposition, Machine Learning, Consistency Based Diagnosis, Fault Identification.

1 Introduction

The diagnosis task has been approached using a wide variety of techniques, being the four main approaches, [Balakrishnan and Honavar, 1998; Narasimhan, 2007]: Knowledge Based—including expert systems,— Case Based , Data Driven — including machine learning,— and Model Based Reasoning. Currently, it seems clear that no single technique is capable to claim its success in every field. Therefore, an increasing number of diagnosis systems have opted for mixed solutions. In this work, we propose a combination of Model Based and Machine Learning methods. Our approach relies primarily upon model-based diagnosis, but it has been enhanced via machine-learning techniques to provide fault identification capabilities.

In the Artificial Intelligence field, the DX community has developed Consistency Based Diagnosis, CBD, as the major paradigm for model based diagnosis [de Kleer et al., 1992]. CBD is able to perform both fault detection and localization with just models for correct behavior, but the absence of fault models knowledge is partly responsible of the low discriminative power that CBD may exhibit [Dressler and Struss, 1996]. Usually, to solve this drawback, knowledge about fault modes is introduced. Different proposals have been made depending on the amount of knowledge used about fault modes. In this work, we have considered the predictive approach, which use models of fault modes to estimate faulty behavior, as in Sherlock [de Kleer and Williams, 1989], GDE++ [Struss and Dressler, 1989] Livingstone [Williams and Nayak, 1996] or TRANSCEND [Mosterman and Biswas, 1999]. Based on such estimation, non-consistent fault modes are rejected. However, adding fault modes increase complexity. For \( N \) components in a system, fault isolation must discriminate among \( 2^N \) modes (ok or fault). Fault identification must discriminate among \( K \) modes (for an average \( K \) behavioural modes). The problem becomes more difficult in presence of dynamics.

To avoid this problem, we propose to use CBD without fault models, and include fault identification knowledge obtained with machine learning techniques.

There are several systems that couple model based diagnosis with machine learning. In [Yu et al., 1999], from the neural network perspective, a typology of approaches is presented. In [Console et al., 2003] a review of the compilation approach is included. Currently, three basic approaches can be found in literature, with some proposals in between: compilation, residual classification and model learning.

The compilation method uses machine learning to induce classifiers that model the relations from symptoms to faults. The main objective of the compilation approach is to improve the efficiency of model-based diagnosis usually to perform on-line diagnosis [Console et al., 2003; Murphy et al., 2006]. Compilation approach may use simulation of the models under relevant faults or may use the model based diagnostic system under relevant observations to compute the faults. In both cases, a set of examples that associate observations...
and faults is generated using the models and the examples are provided as input to a learning algorithm.

The residual classification method uses machine learning to induce classifiers that model the relation from residuals to faults. The main objective of this approach is to improve the robustness of the isolation from residuals. The residuals generated by a model-based diagnosis system and fault information is the input of the learning algorithm. The induced model is used on line for fault isolation to obtain a robust isolation from the residuals [Pernestal et al., 2006; Yu et al., 1999].

The model learning method uses machine learning to induce models of the system. The objective of this approach is to obtain models of complex systems whose behavior is not well known. There are two different approaches: learning models of correct behavior or learning models of faulty behavior. Models of correct behavior are obtained from input output data of normal operation to create a model of the system. This model is used on line to generate residuals. Neural networks have been extensively used for learning models of correct behavior [Patton et al., 1994; Yu et al., 1999], but others approaches are possible, like Bayesian networks [Castañon et al., 2005]. Models of faulty behavior require data of bad operation to create models that perform fault isolation from observations. Examples are the induction of chronicles [Dousson and Duong, 1999] or Hidden Markov Models [Chakrabarti et al., 2007]. Bayesian networks are also a popular method, although usually the structure of the network is given and training data are used to adjust the network parameters [Lerner et al., 2000].

In this work, the compilation approach is used to combine consistency based diagnosis with machine learning techniques, maintaining the soundness of the CBD approach. CBD is in charge of fault detection and localization, while machine learning is used for fault identification. The identification problem is approached as a multivariate time series classification task and time series classifiers are induced off line from simulated data. This approach has been previously tested in [Alonso et al., 2007].

The main contribution of this paper is to apply the structural knowledge provided by the CBD method to define the input-output structure of the classifiers. We propose to use Possible Conflicts, PCs, [Pulido and Alonso Gonza\'lez, 2004] for both Fault Detection and Localization, and system decomposition. We briefly describe the machine learning techniques. Then we explain how PCs decomposition identifies subsystems that provide fault mode selection and attribute selection in a systematic way, and propose to use PCs to define the structure of an ensemble of classifiers. This ensemble can be easily integrated in the CBD cycle without affecting localization soundness. The approach is tested in a simulated scenario and systematically evaluated.

2 Possible Conflicts for On Line CBD and System Decomposition

The computation of possible conflicts is a compilation technique which, under certain assumptions, is equivalent to on-line conflict calculation in the General Diagnostic Engine [de Kleer and Williams, 1987], GDE, or Fault Detection and Isolation using ARRs obtained through structural analysis. A detailed description of consistency based diagnosis with possible conflicts can be found in [Pulido and Alonso Gonz\'alez, 2004].

The main idea behind the possible conflict concept is that the set of subsystems capable to generate a conflict can be identified off-line, in a three steps process:

The first step represents the system as a hyper-graph, 

\[ H_{SD} = \{V, R\} \]

where \( V \) is the set of variables of the system and \( R = \{r_1, r_2, \ldots, r_n\} \) is a family of sub-sets in \( V \), where each \( r_k \) represents a constraint in the model.

The second step looks for minimal over-constrained sub-systems, called Minimal Evaluation Chains (MEC), 

\[ H_{ec} = \{V_{ec}, R_{ec}\} \]

where \( V_{ec} \subseteq V \), \( R_{ec} \subseteq R \). Evaluation chains are necessary conditions for a possible conflict to exist. Additionally, each MEC identifies, by definition, a subsystem of \( H_{SD} \).

In the third step, extra knowledge is added to assure that a MEC, \( H_{ec} \), can be solved using local propagation criterion. If it is possible, a Minimal Evaluation Model (MEM) is defined: 

\[ H_{mem} = \{V_{mem}, R_{mem}\} \]

with \( V_{mem} = V_{ec} \) and \( R_{mem} = \{r_{1,1}, r_{2,1}, \ldots, r_{m,1}\} \). \( r_{ik} \) is a casual constraint obtained assigning a causality to \( r_k \in R_{ec} \). The set of relations of a MEC with at least one MEM is called a possible conflict (PC).

Actually a MEM is a computational model with discrepancy detection capability. Hence, the set of MEMs can be used to perform fault detection. If there is a discrepancy between predictions from those models and current observations, the possible conflict would be responsible for such a discrepancy and must be confirmed as a real conflict. Afterwards, diagnosis candidates are obtained from conflicts following Reiter’s theory.

Additionally, MEMs provide a mean to decompose the original system, \( H_{SD} = \{V, R\} \). Each MEM is uniquely related to one MEC. And each MEC identifies, by definition, a subsystem of \( H_{SD} \). Then, the set of MEMs induces a decomposition on \( H_{SD} \). The decomposition is not exhaustive (some variables and relations may not be included in any subsystem) neither exclusive (some relations and variables may belong to various subsystems), but it is systematic because the algorithms that compute possible conflicts find every MEC and MEM. An important property of this decomposition is that every found subsystem is minimal in the sense that no proper subsystem has discrepancy detection capability. This assures that the decomposition induced by MEMs is unique.

3 Time Series Classifiers for Fault Identification

In this work, fault identification is approached as a problem of multivariate time series classification. This is adequate for those dynamic systems where the set of available observations (measurements and systems parameters and settings) is fixed and no new measurement is available to refine diagnosis candidates. We restrict ourselves to discrete and finite time series of real numbers. Several systems, like industrial continuous processes or autonomous
systems, satisfy these requirements. The historic values of each observable variable may be considered as a univariate time series and the set of historic values of all the variables of interest as a multivariate time series. Therefore diagnosis of past and current faults may be achieved analyzing the past multivariate time series, particularly inducing multivariate time series classifiers. Although machine learning has been widely used for diagnosis problems, to the best of our knowledge, few authors formulate the problem as a multivariate time series classification [Roverso, 2003; Alonso et al., 2007].

To test the effect of system decomposition, we have selected six machine learning techniques: Decision Trees (DT), Naive Bayes (NB), Support Vector Machines (SVM) (with kernels linear and perceptron), Nearest Neighbor (k-NN), and Stacking Nearest Neighbor (Stack-k-NN).

DT, NB, SVM and k-NN are standard machine learning techniques. Particularly, k-NN using Dynamic Time Warping, DTW, as a dissimilarity measure behaves reasonably well for most univariate problems [Keogh and Ratanamahatana, 2005]. Although DTW may be extended to the multivariate case, it does not behave well for this problem. As an alternative approach, Stack-k-NN is proposed in [Alonso et al., 2007]. This is an ensemble method that obtain multivariate time series classifiers from k-NN univariate time series classifiers.

Stack-k-NN is a variant of the Stacking ensemble method. Stacking is characterized by two classification levels: level 0, which produces different classifiers for the same data set, and level 1, which tries to generalize from level 0 classifications. Level 1 generalization is accomplished by inducing another classification model from the same data set, once it has been processed by the level 0 classifiers [Wolpert, 1992].

Stack-k-NN use the Stacking configuration shown in Figure 1. If the original multivariate time series has n variables, it is split into its n components to obtain n univariate times series. For each time series, k-NN algorithm is used. The output of these level 0 classifiers plus class information are arranged in a vector and used to train the level 1 classifier through a ten folds stratified cross validation. As level 1 learner we have chosen Naive Bayes.

Experimental results have shown that Stack-k-NN outperforms the five standard machine learning algorithms considered on various data sets.

When necessary, the machine learning algorithms have been adapted to provide class confidences instead of assigning a single class. This feature will be used to combine the classifiers outputs when decomposition selects more than one classifier.

4 Attribute and Class Selection via Possible Conflicts

In previous works, knowledge about the system to be diagnosed has not been used to decide the input-output structure of the classifiers. A single global classifier, Classifier, was constructed. Let n be the number of available observations. Then, the input space of the classifiers is the multivariate time series space

\[ I = TS_1 \times TS_2 \times \ldots \times TS_n \]

with \( TS_i \) the time series space associated to observation i. Respecting fault modes, we assume system detectability, and only consider fault modes that may be detected by some possible conflict, according to the fault signature matrix of the system. Let’s \( C_i \) be the set of fault models associated to \( PC_i \). Then, the set of classes, C, is defined as \( C = \bigcup_i C_i, i = 1, 2, m \), with m the number of PCs. Therefore, the classifier Classifier, applies I into C:

\[ \text{Classifier} : I \rightarrow C \]

Although Stack-k-NN seems to behave satisfactory with this simple setting for a small size problem, it is not to be expected that the method may scale up to systems with hundreds of observations and fault modes.

On practical applications, a great effort is usually done to analyze the characteristics of the problem. It is well known [Frank and Kramer, 2004; Li et al., 2005; Rokach, 2006] that the induction of classifiers is increasingly difficult with the number of classes to consider and with the number of attributes; particularly harmful is the presence of irrelevant and/or redundant attributes. A lot of work has been done to automate this analysis, specially in the field of attribute selection. However, it is still claimed that a thorough knowledge of the problem and the help of problem experts is necessary in all but simple applications [Witten and Frank, 2005].

One obvious way of simplifying the problem is decomposing the physical system, looking for subsystems where the space I has lower dimension and the set C has lower cardinality. The Possible Conflict approach provides a systematic method to decompose a physical system.

4.1 Possible Conflict Decomposition: Class Decomposition

We have shown in section 2 that the set of MEMs induce a unique decomposition on the system. Because each MEM, is only sensible to the mode faults of \( C_i \), we have that \( |C_i| \leq |C| \). This provides a first decomposition step that reduces the number of classes of a classifier. If we have m Possible Conflicts, the we can replace the single global classifier, Classifier, by m local classifiers, Classifier – \( C_i \), where the suffix \( C_i \) indicates that decomposition has been applied.
to class selection and subindex $i$ identify the PC used. The structure of these classifiers is:

$$\text{Classifier} - C_i : I \rightarrow C_i$$

These local classifiers, $\text{Classifier} - C_i$, still work on the same input space, but have the potential to work with less number of classes, because Possible Conflicts are minimal.

### 4.2 Possible Conflict Decomposition: Attribute and Class Decomposition

The decomposition may be carried out a further step, to the level of the $\text{MEM}$ subsystem. Now, the $\text{MEMs}$ project the input space $I$ into $m$ subspaces $I_i$, with $m$ the number of $\text{MEMs}$ and $I_t$ the space of time series associated to the observations of $\text{MEM}_i$, that is, the observations of $\text{V}_{mem_i}$. The projections are not exclusive and several subspaces may share several dimensions, that is, observations. Note also that some dimensions of $I$ may not be projected in any subspace.

With this ultimate level of decomposition, the original global classifier, $\text{Classifier}$, is replaced by $m$ local classifiers, $\text{Classifier} - AC_i$, where the suffix $AC$ indicates that decomposition has been applied to classes and observations (attributes). The structure of these classifiers is:

$$\text{Classifier} - AC_i : I_i \rightarrow C_i$$

### 4.3 Possible Conflict Decomposition: Ensemble of Classifiers

During the identification stage, several classifiers might be invoked for the same fault, as explained in the next section. Actually, the $m$ local classifiers may be considered as an ensemble of classifiers. We define $\text{Ensemble-C}$ (resp. $\text{Ensemble-AC}$) as the ensemble of $m$ local classifiers $\text{Classifier} - C_i$ (resp. $\text{Classifier} - AC_i$). Then, a mechanism is needed to combine the class prediction of the local classifiers. We use a simple average scheme. The local classifiers assign a probability to each class of their output domain and a null probability to the remainder classes. These probabilities are averaged and provided as the output of the ensemble.

By definition, the family of $C_i$ is a coverage of $C$, although not a mutually exclusive one, because it is possible to have $C_i \cap C_j \neq \emptyset$ for some $i \neq j$. Hence some degree of overlapping among the set of classes is to be expected. There is also some overlapping among the dimensions of the input space of the classifiers when attribute selection is applied. From the machine learning point of view, this is a desirable property: we may have several classifiers providing different confidence to the same classes. This has the potential to generate diversity, which is a much looked after property on the ensemble methods.

### 5 On Line Fault Detection, Isolation and Fault Identification

Consistency-based diagnosis automatically provides fault isolation based on fault detection results. Using possible conflicts, CBD can be easily done without on-line dependency recording. The proposed diagnosis algorithm will incrementally generate the set of candidates consistent with observations. In the off-line stage, the system model is automatically analyzed to find out every possible conflict, $PC_i$, and its associate executable model, $\text{MEM}_i$.

In the on-line stage, we perform a semi-closed loop simulation with each executable model $\text{MEM}_i$:

1. **repeat**
   - (a) simulate($\text{MEM}_i; OBS_{\text{MEM}_i}$) $\rightarrow$ $\text{PRED}_{\text{MEM}_i}$.
   - (b) if $|\text{PRED}_{\text{MEM}_i} - OBS_{\text{MEM}_i}| > \delta_{\text{MEM}_i}$, confirm PC$_i$ as a real conflict.
   - (c) update(set of candidates, set of activated PCs)

2. **until** every PC$_i$ is activated or time elapsed.

Where $OBS_{\text{MEM}_i}$ denotes the set of input observations available for $\text{MEM}_i$; $\text{PRED}_{\text{MEM}_i}$ represents the set of predictions obtained from $\text{MEM}_i$; $OBS_{\text{MEM}_i}$ denotes the set of output observations for $\text{MEM}_i$; and $\delta_{\text{MEM}_i}$ is the maximum value allowed as the dissimilarity value between $OBS_{\text{MEM}_i}$ and $\text{PRED}_{\text{MEM}_i}$.

This algorithm can be easily modified to include fault identification information from time series classifiers.

Let’s call $\text{Ensemble}$ to any of the ensembles of local classifiers introduced in section 4, $\text{Ensemble-C}$ or $\text{Ensemble-AC}$.

Let $\text{Ensemble}(t, \text{set of activated PCs})$ denote an invocation of $\text{Ensemble}$, with a fragment of series from $t$ to the min(current time, $t$ + maximum series length) and information of the activated PCs. This information is needed to select the local classifiers that have to be used.

With this notation, the integration of the fault mode knowledge in the consistency based diagnosis cycle may be simply stated. Just add:

(d) $\text{Ensemble}(t_0, \text{set of activated PCs})$

to the on-line simulation loop, with $t_0$ the starting time of the series, prior to the first conflict confirmation. The new algorithm provides fault isolation using CBD, and fault identification information, ordering fault modes according to the confidence assigned to them by the ensemble of classifiers.

Note that the isolation step is sound in the sense that fault isolation is purely consistency based. Moreover, the local classifiers, by construction, only returns fault modes associated with the activated PCs. Hence, although fault mode assignment departs from the consistency based framework, it is always coherent with the fault isolation stage, in the sense that a fault mode is not selected if the component affected by the fault is not a candidate.

### 6 A Case Study

#### 6.1 The System to be Diagnosed

For this work, we have used the laboratory scale plant shown in figure 2. Although a laboratory plant, its complexity is comparable to the one encountered in several subsystems of real 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 $P_1, P_2$ to keep the level of $\{T_1, T_2\}$ close to the specified set point. To control temperature on tanks $\{T_2, T_3\}$ we use two resistors $\{R_2, R_3\}$, respectively.

In this plant we have eleven different measurements: levels of tanks $T_1$ and $T_4$ $\{LT01, LT04\}$, the value of the PID...
6.2 Possible Conflicts for the System

We have used common equations in simulation for this kind of process:

1. \( t_{dm} \): mass balance in tank \( t \).
2. \( t_{EB} \): energy balance in tank \( t \).
3. \( t_{fs} \): flow from tank \( t \) to pump.
4. \( t_{fs} \): flow from tank \( t \) through a pipe.
5. \( r_{f} \): resistor failure.

Based on these equations we have found the set of possible conflicts shown in table 1. In the table, second column shows the set of constraints used in each possible conflict, which are minimal with respect to the set of constraints. Third column shows those components involved. Fourth column indicates the estimated variable for each possible conflict. We have considered the fourteen fault modes shown in table 2.

### Table 1: Possible conflicts found for the laboratory plant; constraints, components, and the estimated variable for each possible conflict.

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Components</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( PC_1 )</td>
<td>( T_{dm}, T_{d1}, T_{d2} )</td>
<td>( T_1, P_1, P_2 )</td>
</tr>
<tr>
<td>( PC_2 )</td>
<td>( T_{d1}, T_{d2} )</td>
<td>( T_1, T_2, P_1 )</td>
</tr>
<tr>
<td>( PC_3 )</td>
<td>( T_{d1}, T_{d2}, T_{d3} )</td>
<td>( T_1, P_1, T_2, P_2 )</td>
</tr>
<tr>
<td>( PC_4 )</td>
<td>( T_{d1}, T_{d2}, T_{d3} )</td>
<td>( T_1, P_1, P_2, T_3 )</td>
</tr>
<tr>
<td>( PC_5 )</td>
<td>( T_{d1}, T_{d2}, T_{d3} )</td>
<td>( T_4 )</td>
</tr>
<tr>
<td>( PC_6 )</td>
<td>( T_1, P_1, P_2 )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2:** The diagram of the plant.

It should be noticed that these are the fault mode classes which can be distinguished for fault identification. In the localization stage, the following pair of faults \( \{ f_{11}, f_{12} \}, \{ f_{21}, f_{22} \} \), and \( \{ f_{10}, f_{13} \} \) can not be separately isolated.

The fault signature matrix provides the class decomposition. For each \( PC_i \), \( C_i \) includes the fault modes marked \( i \) on the corresponding row. For instance \( C_5 = \{ f_{11}, f_{12}, f_{13} \} \).

### 6.4 Attribute and Class Decomposition

The input an output observations of each possible conflict, obtained from each \( MEM \), are shown in table 4. Now, each subspace \( I_i \) includes the observations marked \( I_i \) on the corresponding column. For instance \( I_3 = T_{FSFT02} \times T_{SFT01} \times T_{S_{LC01}} \times T_{S_{TT02}} \).

### 7 Experimental Evaluation

Due to the cost of obtaining enough data for a fourteen classes classification problem from the laboratory plant, we have resorted to a detailed, non linear quantitative simulation of the plant. We have run twenty simulations for each set of constraints in each fault mode. Each simulation lasted 900 seconds. We randomly generate the fault magnitude and its origin, in the interval \([180, 300]\). The sample rate is 3 seconds. Since we just have eleven observations, each simulation will provide eleven series of three hundred numeric elements.

For complexity reasons, we only consider single faults. We also have assumed that the system is in stationary state before the fault appears.

It is not easy to systematically evaluate the behavior of the diagnosis algorithm presented in section 5. Hence a simplifi-
Table 4: PCs and their associated observations

<table>
<thead>
<tr>
<th>FT 01</th>
<th>PC 1</th>
<th>PC 2</th>
<th>PC 3</th>
<th>PC 4</th>
<th>PC 5</th>
<th>PC 6</th>
<th>PC 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT 02</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT 03</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT 04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT 01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT 04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LC 01</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC 04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT 02</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TT 03</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The conclusion is that applying class and feature selection, 1-NN and Stack-1-NN are the preferred methods. If we take into account the cost of training Stack-1-NN, 1-NN is an excellent option. Class decomposition seems to be desirable. The effect of attribute decomposition is not so clear, although significantly improves the accuracy of the most precise classifiers with 30% of the series, which is important for early fault identification. And additional benefit of attribute decomposition, not considered in this evaluation, is that it may reduce by an important factor the training time respect to class decomposition, because all the local classifiers works on a smaller input space if attribute decomposition is applied.

8 Related Work

System decomposition is generally acknowledged as a powerful method to reduce complexity. However, few works address this problem in a systematic and automatic way in the model based and data mining communities.

In the model based community, a similar idea to the Possible Conflict decomposition, Decompositional Model Based Learning, was first proposed in [Williams and Millar, 1998] for a different task: the use of dissents to decompose complex system to facilitate parameter estimation. The main difference being that dissents computation is not complete, because cyclic configurations are avoided. Additionally, a concept like Class decomposition is not applicable because of the different nature of the task.

Possible conflict has also been proposed to identify the structure of state observers from MEMs in [Pulido et al., 2007]. Implicitly, it is performed the same system decomposition that we exploit in this work. Again Class decomposition do not apply.

The machine learning community has dedicated a great effort to the field or feature (attribute) selection [Witten and Frank, 2005]. Nevertheless, the approach is quite different because the objective is to find general methods that apply to data sets. To the best of our knowledge, there is no work that exploit models of the system to relate classes and attributes.

Recently [Rokach, 2006] has proposed a decomposition methodology for classification task. Attribute decomposition, where the available set of attributes is projected on different subspaces, is given a relevant role. Similar ideas are used for generating ensemble classifiers [Ho, 1998]. However, these methods are oriented towards data set manipulation, and no knowledge of the system is used.

[Rokach, 2006] also considers intermediate concept decomposition with two variants: concept aggregation and functional decomposition. Concept aggregation may group classes into subclasses or be oriented towards transforming multiclass problems into binary classification problems [Dietterich and Ghulum, 1995]. However, no knowledge of the problem is used, unless manually introduced. Functional decomposition is closer in spirit to our approach. It recursively decompose the target concept in sub concepts and subsets of attributes. The decomposition may be autonomous, based on
Table 5: Accuracies and deviations of the methods. The symbol “•” indicates that the result is significantly worse than the result for the last method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Attribute</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>70%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>No</td>
<td>-</td>
<td>65.95</td>
<td>94.76</td>
<td>90.95</td>
<td>94.29</td>
<td>93.81</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>Yes</td>
<td>No</td>
<td>84.52</td>
<td>95.71</td>
<td>94.76</td>
<td>94.76</td>
<td>93.81</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>Yes</td>
<td>Yes</td>
<td>86.67</td>
<td>94.52</td>
<td>94.76</td>
<td>94.29</td>
<td>93.81</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>No</td>
<td>-</td>
<td>57.86</td>
<td>88.10</td>
<td>91.90</td>
<td>92.38</td>
<td>82.86</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Yes</td>
<td>No</td>
<td>85.95</td>
<td>95.00</td>
<td>97.14</td>
<td>97.38</td>
<td>97.86</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Yes</td>
<td>Yes</td>
<td>85.00</td>
<td>98.33</td>
<td>98.33</td>
<td>98.33</td>
<td>98.57</td>
</tr>
<tr>
<td>SVM linear</td>
<td>No</td>
<td>-</td>
<td>42.62</td>
<td>83.10</td>
<td>87.14</td>
<td>89.52</td>
<td>94.05</td>
</tr>
<tr>
<td>SVM linear</td>
<td>Yes</td>
<td>No</td>
<td>74.29</td>
<td>93.33</td>
<td>97.14</td>
<td>98.81</td>
<td>99.52</td>
</tr>
<tr>
<td>SVM linear</td>
<td>Yes</td>
<td>Yes</td>
<td>75.71</td>
<td>91.19</td>
<td>96.17</td>
<td>91.43</td>
<td>92.14</td>
</tr>
<tr>
<td>SVM perceptron</td>
<td>No</td>
<td>-</td>
<td>50.24</td>
<td>82.62</td>
<td>86.90</td>
<td>88.57</td>
<td>88.81</td>
</tr>
<tr>
<td>SVM perceptron</td>
<td>Yes</td>
<td>No</td>
<td>87.86</td>
<td>98.10</td>
<td>99.05</td>
<td>98.57</td>
<td>98.57</td>
</tr>
<tr>
<td>SVM perceptron</td>
<td>Yes</td>
<td>Yes</td>
<td>86.67</td>
<td>97.14</td>
<td>98.33</td>
<td>98.33</td>
<td>99.05</td>
</tr>
<tr>
<td>T-NN, DTW</td>
<td>No</td>
<td>-</td>
<td>48.81</td>
<td>84.52</td>
<td>85.00</td>
<td>86.67</td>
<td>87.62</td>
</tr>
<tr>
<td>1-NN, DTW</td>
<td>Yes</td>
<td>No</td>
<td>83.10</td>
<td>94.76</td>
<td>95.00</td>
<td>95.24</td>
<td>96.90</td>
</tr>
<tr>
<td>1-NN, DTW</td>
<td>Yes</td>
<td>Yes</td>
<td>92.62</td>
<td>98.33</td>
<td>98.57</td>
<td>98.57</td>
<td>99.05</td>
</tr>
<tr>
<td>Stack-1-NN+NB</td>
<td>No</td>
<td>-</td>
<td>63.81</td>
<td>96.67</td>
<td>97.86</td>
<td>98.10</td>
<td>99.05</td>
</tr>
<tr>
<td>Stack-1-NN+NB</td>
<td>Yes</td>
<td>No</td>
<td>89.05</td>
<td>98.33</td>
<td>99.05</td>
<td>98.81</td>
<td>99.76</td>
</tr>
<tr>
<td>Stack-1-NN+NB</td>
<td>Yes</td>
<td>Yes</td>
<td><strong>93.10</strong></td>
<td><strong>99.52</strong></td>
<td><strong>99.76</strong></td>
<td><strong>99.52</strong></td>
<td><strong>99.76</strong></td>
</tr>
</tbody>
</table>

analysis of the data and partitions measurements, or supervised, with guidance from the user [Frank and Kramer, 2004; Zupan et al., 1999].

9 Conclusions and Further Work

This work has shown how Possible Conflicts technique may be used to decompose a system, using the computational models associated to each Possible Conflict, that is, its Minimal Evaluation Model.

This decomposition may be used to define the input-output structure of induced classifiers. The decomposition defines the structure of an ensemble of classifiers that may be integrated on line in the Consistency Based Diagnosis cycle, providing fault identification information. The integration relies on the compilation approach of Machine Learning and Model Based Diagnosis.

The proposal has been systematically evaluated on a simplified scenario on various machine learning methods. Experimental results show that Class decomposition systematically improves the accuracy of the classifiers. The effect of Attribute decomposition is not so consistent, although significantly improves the accuracy of the most precise classifiers at the first stages of the diagnosis processes.

The major contribution of this work is that it provides a systematic approach to define the structure of an ensemble of classifiers for fault identification form an abstract model of the system, using Possible Conflict decomposition.

Future work requires a thorough evaluation of the proposal on the same pilot plant used in this paper. Data from real faults have been collected. Also, additional simulation data must be generated, to evaluate the behavior of the classifiers independently of the possible conflicts detection and isolation accuracy.

It would be interesting to evaluate the proposal with different levels of decomposition, because grouping attending fault modes and/or observations might improve the classifiers accuracy.

References


Distributed Consistency-Based Diagnosis without Conflicts

Vincent Armant and Philippe Dague and Laurent Simon
Univ. Paris Sud (LRI)& CNRS - INRIA (Saclay)
4 rue Jacques Monod, Parc Club 91400 Orsay

Abstract

Some methods exist to locate errors and incorrect answers in a distributed framework, where all peers work together for the same purpose, under the same protocol. For instance, one may limit them by rep-lication of data and processes among the network. However, with the emergence of web services, the willing for privacy, and the constant growth of data size, such a solution may not be applicable. For some problems, failure of a peer has to be detected and located by the whole system. In this paper, we propose an approach to diagnose abnormal behaviors of the whole system by extending the well known consistency-based diagnosis framework to a fully distributed inference system, where each peer only knows the existence of its neighbors. Contrasting with previous works on model-based diagnosis, our approach computes all minimal diagnoses in an anytime way, without needs to get any conflict first.

1 Introduction

Model-Based Diagnosis has been introduced in the late eighties by [Reiter, 1987; de Kleer and Williams, 1987], and has since been widely used in many successful works. With this formalism, a logical theory describes the normal (and, optionally, abnormal) behavior of a physical system, and inconsistent observations are used to derive hypotheses over components reliability (called diagnoses), that explains failures. Even if stronger logic may be used, it is often the case where propositional logic is chosen to model the system. In this context, diagnosing the system with respect to observations can be expressed as a classical – and heavily studied – knowledge based compilation problem : restricted prime im-plicants [Darwiche and Marquis, 2002].

Recent years have seen an increasing number of AI works pushing forward the power of Distributed system, for instance by adding semantic layers [Adjiman et al., 2005]. In such networks, all systems (or “peers”) are running the same algo-rithm, and are working for the same purpose. The framework may however describe two kinds of settings. One which allows any peer to communicate with any other peer (generally by means of distributed hash tables, [Stoica et al., 2001]) or the other where peers only know their neighbors, which is closer to social networks, circuits, and web services compo-sition. In the latter formalism, reasoning is based on the declaration of logical equivalence of variables between peers (the shared variables), which locally defines subsystems acquaintances.

In this paper, we investigate the problem of diagnosing distributed systems defined by peers acquaintances. Each peer only knows its neighborhood, and has a logical model of its normal and abnormal behavior with respect to its observations and its shared variables with its acquaintances. The challenging problem is to build a set of global diagnoses of the whole system. Our solution directly computes diagnoses (including all minimal ones, either for set inclusion or for cardinality) without conflicts analysis, a very hard task which is generally the first step – and the first bottleneck – of all previous model-based diagnoses engines, even when efficient algorithms are used [Simon and del Val, 2001].

In our approach, we focus on “static” settings of distributed systems, in order to easily ensure that diagnoses and observa-tions are consistent. If the static behavior is not possible in a fully peer-to-peer setting, it is more realistic in a distributed setting, for instance web services composition, embedded cir-cuits, and social networks. In many cases, additional layers, like memory of past events and counters, can even simulate the “static” hypothesis.

In the next section, we recall the principles of model based diagnosis and introduce our notations. In section 2, we extend model-based diagnoses to formulas in Disjunctive Normal Form and, section 3, we introduce our fundations of distributed reasoning for diagnoses. In section 4, we present the distributed algorithm and then we report related work and conclude.

Example 1 (Three steps web-payment certification) We illustrate the paper by an example of a web-payment certification, figure 1. The order validation service (OVS) asks for a shipping service (ES) for a hire purchase approval (hpPurch). In order to maximize its sales opportunity, (ES) waits for the customer bank approval (bkAprvl) or a loan agency approval (laAprvl). The bank hire purchase service (HPS) and the loan agency service (LAS) both check the customer credit card validity (valCC) by a call to the credit card service (CCS). In the following, we restrict the system to (HPS) and will refer to its “global description” as the conjunction of the Transaction Approval (TA), the Solvability Checking (SC) and the Option Checking (OC).
2 From CNF Diagnosis to DNF Diagnosis

We assume familiarity with the standard literature on propositional reasoning and resolution. A literal is a variable or its negation \( \neg v \). Given a set \( L \) of literals, we denote by \( L^c \) the set of its opposite literals. A Conjunctive Normal Form (CNF) formula is a conjunction of disjunctions of literals. A Disjunctive Normal Form (DNF) formula is a disjunction of conjunctions of literals. For simplicity, we identify a formula with a set of sets of literals. We denote by \( T^c \) (resp. \( T^v \)) the set of sets of literals corresponding to a CNF (resp. DNF).

A model is a set of literals that, when assigned to true, allows the evaluation of a given formula to true. We say that a formula \( f \) is satisfiable (or consistent), denoted by \( f \models \bot \), if there exists a model for \( f \). Let \( f_1 \) and \( f_2 \) be two formulas, if all the models of \( f_1 \) are also models of \( f_2 \), then \( f_1 \models f_2 \). A literal \( \ell \) is called an implicat of \( f_2 \) if \( f_2 \models \ell \). A correct description of \( f_2 \) with respect to the order relation induced by inclusion of sets of literals is called a prime implicates (resp. prime implicants) of \( f_2 \). The set of prime implicants is expressed in CNF, whereas the set of prime implicates is expressed in DNF.

A formula \( f \) and a subset \( V \) of its variables, the restriction of \( f \) on \( V \) is denoted by \( f|_V \) and corresponds to recursively apply on \( f \) the Shannon decomposition operator on all variables \( x \) of \( f \) that do not appear in \( V \). This operation, known as forgetting in Knowledge Compilation, is well known to be a NP-Hard problem. However, when \( f \) is expressed as a DNF, the restriction operator is simply a vocabulary restriction of all products of \( f \). The restriction of \( f|_V \) on a set of literals \( L \) can be defined as \( f|_V([L]) = \{ I^c \mid \exists I \in T^v \text{ s.t. } I^c = I \cap L \} \), which is no more a hard task.

2.1 Centralized Model-Based Diagnosis

Like many other works, we adapt the model-based diagnosis framework from [de Kleer and Williams, 1987; de Kleer et al., 1992] to the propositional case. Initially, an observed system is a triple \((SD, COMPS, OBS)\) where \( SD \) is a first order logical formula describing the system behavior, \( OBS \) is a formula describing the observations (that boils down frequently to values assignment to observable variables) and \( COMPS \) is the set of monitored components, that appear as arguments of the predefined predicate \( Ab() \) in \( SD \) (\( Ab(C_i) \) denoting that component \( C_i \) is abnormal). In propositional logic, we may merge the whole into a single theory \( T \), with the naming convention : all variables \( okC_i \) (called mode variables) encode the correct behavioral modes of the components \( C_i \), i.e. \( \neg Ab(C_i) \). We note \( F \) the set of negative mode literals \( \{ \neg okC_1, ..., \neg okC_n \} \) representing faulty components. For a (boolean) observable coded by a variable \( v \), the elementary observation \( v = a \) is coded by \( v \) if \( a \) equals 1 and \( \neg v \) if \( a \) equals 0.

Example 2 (Modeling the system) A correct behavior of TA (\( okTA \)) will approve a hire purchase (\( bkAprvl \)) if the customer is solvent (\( solv \)) and fulfills the condition (\( eOpt \)) of OC. The rule for TA is rewritten as \( f(TA) = f(TA) \models (okTA \rightarrow (solv \land eOpt \land bkAprvl)) \). A normal functioning of SC (\( okSC \)) will consider a customer solvent (\( solv \)) if he does not exceed his overdraft limit (\( \neg exOvLine \)). We obtain \( f(SC) = (okSC \rightarrow (\neg exOvLine \land solv)) \). A correct behavior of OC (\( okOC \)) will satisfy (\( eOpt \)) if the customer asked for hire purchases by internet (\( ePurch \)) and his credit card is valid (\( valCC \)). There are only two possible failures for OC : when \( ePurch \) keeps its default value whereas the customer asked for this option, and when the customer card is believed invalid whereas it is. The Option Checking system can thus be encoded by \( f(OC) = (okOC \rightarrow (ePurch \land valCC \land eOpt)) \land (\neg okOC \rightarrow \neg valCC \lor \neg ePurch) \). The behavior of HPS is the conjunction \( f(HPS) = (f(SC) \land f(SC) \land f(TA)) \). The theorem 3 of [de Kleer et al., 1992] states that the minimal diagnoses are the prime implicants of the conjunct of minimal conflicts, where the minimal conflicts (called minimal conflict sets in [Reiter, 1987] and minimal positive conflicts in [de Kleer et al., 1992]) are the prime implicates of the formula \( SD \land OBS \), restricted to the mode literals in \( F \). Intuitively, a minimal conflict refers to a set of components containing at least a faulty one. Minimal diagnoses are thus the smallest conjunctions of faulty components that can explain all the faults, according to observations.

Definition 1 (Minimal Diagnoses) Let \( T \) be the theory that describes an observed system, \( F \) the consistent set of all negative mode literals of the system.

\[ \Delta \subseteq F \text{ is a diagnosis for } T \text{ iff } T \cup \Delta \cup \{ F \setminus \Delta \} \models \bot \]

We write \( \text{Diag}(T) \) the set of diagnoses of \( T \) and \( \text{min}_\Delta(\text{Diag}(T)) \) the set of its minimal diagnoses. Intuitively, this definition states that, given any minimal diagnosis \( \Delta \) of the observed failure, one may suppose that all components \( C' \) that do not appear in \( \Delta \) are correct. We may also notice that, because we can restrict the set of possible failures \( (\neg okOC \rightarrow \neg valCC \land ePurch) \) in the previous example), a diagnosis may not be extended by supposing all components \( C' \) incorrect (two negative mode literals extending a diagnosis may be however mutually exclusive).

Example 3 (Conflicts and Diagnoses on Scenario 1) Let’s suppose the following scenario : the bank approved a hire purchase for an operation whereas the customer exceeds his overdraft limit. He had a valid credit card but asked to stop internet purchasing. In this case the bank service does not fulfill its expected behavior.

We thus look for the minimal subsets of \{ TA, OC, SC \} that may be faulty. This will be expressed by minimal conjunctions of literals from \{\( \neg okTA, \neg okOC, \neg okSC \}\}, which are consistent with the formula \( f(HPS) \) and the observations \( (exOvLine, valCC, \neg ePurch, bkAprvl) \). The minimal conflicts are \( (okSC \land okTA) \) and \( (okOC \land \neg ovf \land \neg okTA) \). The minimal diagnoses that satisfy these conflicts are : \( (\neg okTA) \) and \( (okSC \land \neg okOC) \).
Most of previous work on diagnosis compute first the set of conflicts, restricted to mode literals. Then, only when this first stage is over, diagnoses can be computed. These methods are hopeless for building an anytime diagnostics engine as all conflicts have to be known before the first diagnoses can be returned. They are nevertheless motivated by the fact that models of real-world are supposed to be close to CNF. If needed, new variables are usually added to practically contain the potential blow-up when translating to CNF. However, a DNF representation of a circuit is from great interest: If we ensure that the set of mode variables is consistent, which means that no variables are both positively and negatively in $F$, then, if $T'$ is the description of an observed system, each product of the restriction $T'|_{\{F\}}$ is a diagnosis (not necessarily minimal).

**Lemma 1** Let $T'$ be a DNF description of an observed system, and $F$ a consistent set of mode literals, then

$$\forall I \in T', I|_{\{F\}} \in \text{Diag}(T)$$

**Sketch of Proof** For each $I \in T'$, $I$ is an implicant of $T$ which is trivially consistent with $T$. Let us consider ${I|_{\{F\}}}$, which does not contain any literals from $T|_{\{F\}}$. Thus, $T \cup I|_{\{F\}} \cup F \setminus I|_{\{F\}}$ is consistent with $T$ and by definition $I|_{\{F\}}$ is a diagnosis.

Consequently, if we compute at least one implicant of $T$, we obtain at least one diagnosis without waiting for conflicts. In practice, our requirement about DNF representation of $T$ can be weakened without loss: implicants can be incrementally computed by an efficient SAT Solver or even deduced from a compact, compiled representation of $T$. Let us consider a DNF description of an observed system containing mode variables.

**Theorem 1** Let $T$ be the description of an observed system, and $F$ a consistent set of mode literals:

$$\text{min}_c(\text{Diag}(T)) = \text{min}_c(T'|_{\{F\}})$$

**Sketch of Proof 1** Let $\Delta$ be a minimal diagnosis, by the definition $1$, $\Delta \cup \{F \setminus \Delta\}$ is consistent with the observed system. Moreover, for any DNF formula of the system, there exists an implicant $I$ consistent with $\Delta \cup \{F \setminus \Delta\}$. Since $I$ is consistent with $\Delta \cup \{F \setminus \Delta\}$ we have $I|_{\{F\}} = \emptyset$ and thus $I|_{\{F\}} = I|_{\{\Delta\}}$. Because we know that $I|_{\{F\}}$ is a diagnosis and $\Delta$ is a minimal one we have $I|_{\{\Delta\}} = \Delta$.

2) Let $I|_{\{F\}} \in \text{min}_c(T')$. $I|_{\{F\}}$ is a diagnosis, suppose that it is not a minimal one. Then there exists a diagnosis $\Delta, \text{s.t.} \Delta \subset I|_{\{F\}}$. Consequently, there exists $l$ in $I|_{\{F\}}$ s.t. $l$ is not in $\Delta$. In this case $\Delta \cup \{F \setminus \Delta\} \cup I$ is contradictory. But since $\Delta \cup \{F \setminus \Delta\}$ is consistent with $T'$, then there exists $I' \neq I$ s.t. $\Delta \cup \{F \setminus \Delta\} \cup I'$ is consistent and $I'|_{\{F\}} \subseteq \Delta$. We deduce that $I'|_{\{F\}} \subseteq \Delta \subseteq I|_{\{F\}}$, It is the contradiction with the fact that $I|_{\{F\}} \in \text{min}_c(T'|_{\{F\}})$.

**Example 4 (Finding Diagnoses in DNF)** Let $F_{\text{HBS}} = \{\neg \text{okTA}, \neg \text{okOC}, \neg \text{okSC}\}$ be the set of mode literals and $T_{\text{HBS}}$ the DNF formula of the description with observations.

$$T_{\text{HBS}} = \{\neg \text{okTA} \land \neg \text{okOC} \land \neg \text{okSC}\}$$

By the lemma 1 we know that each implicant contains a diagnosis. By the theorem 1 we states that all DNF description of the observed system contains the set of minimal diagnoses. Now, suppose that we monitor and diagnose a distributed system consists of diagnosis engine which gradually compute local implicants from the monitored subsystem. A consistent composition of locals implicants from each diagnosis engine is also an implicant for the global system. We note that, as soon as each diagnosis engine return it first implicant, we can start the composition task. In the next section we precise the notion of distributed system which differ from the usual notion of system in Diagnosis by taking into account the shared and local acquaintance of a subsystem. We take advantage of this characterization for forgetting symbols and optimize the composition task.

### 3 Direct Diagnosis of Distributed Settings

We formalize our distributed model based diagnosis framework by means of Peer-to-Peer Inference Systems (P2PIS) proposed by [Halevy et al., 2003], and extended in [Adjiman et al., 2005] for distributed reasoning. In a P2PIS, a so-called “inference peer” has only a partial knowledge of the global network (generally restricted to its acquaintance) and may have local information, only known by itself. In our work, an inference peer will for instance model the expected behavior of a real peer, a web service, or a subcircuit, of a distributed system. Let us denote by $T$ the description of the global observed system. $T$ is the (virtual) conjunction of all local theories $T_p$ of peers $p$. Of course, in our framework, $T$ will never be explicitly gathered and privacy will be ensured on local knowledge. $T$ is built on the global vocabulary $V$ (excluding mode variables), which can be partitioned into shared variables $Sh$ and local variables $Loc$.

$$\text{Loc} = V \setminus Sh$$

In addition to this partition, we have to add mode variables in order to be able to diagnose the system. We denote by $F$ the set of all mode variables of the system. Obviously in order to build global diagnoses, exchange of mode variables between peers has to be possible. Thus, the network will allow formulas build on variables from $SH \cup F$ to be sent from peer to peer. We denote by $V_p, Sh_p, Loc_p, F_p$ the vocabulary, the shared variables, the local variables and the mode variables symbols of any inference peer $p$.

#### 3.1 A network of DNF models

In the previous section, we assumed that we were able to work directly on the DNF of $T$. Because $T$ here is a conjunction of formula, we may push this hypothesis to all $T_p$. If the
first hypothesis may not be considered as a realistic one, at te opposite small peers will admit relatively small DNF en-
coding, and thus the second one is of a practical interest. If all Tp are in DNF, then writing T in DNF can be done by the
distribution property of ∧ over ∨. More formally, we use the
following operator for this purpose :

**Definition 2 (Distribution (⊗))**

\[ T'_1 \otimes T'_2 = \{ i_1 \land i_2 | i_1 \in T'_1, i_2 \in T'_2, i_1 \land i_2 \neq \perp \} \]

One may notice that inconsistent products are deleted, and, if the result is minimised, then this operator is exactly the
clause-distribution operator [of Simon and del Val, 2001], but applied to DNF and products.

Because of privacy, and for efficiency purpose, let us introduce the following lemma stating that instead of distributing all theories before restricting the result to mode variables, one may first restrict all theories to shared and mode variables without loss.

**Lemma 2** Let \( T^\forall \) be a description of an observed P2P system, \( F \) a consistent set of mode literals :

\[ (\otimes T^\forall_p)(|sh,F) = (\otimes T^\forall_p)||sh,F_p) \]

**Sketch of Proof** Let \( I \) (resp. \( I' \)) \( \\beta \leftarrow \\forall \) \( \\beta_{\forall \leftarrow I'} \), local symbols from \( I \) does not appear in \( I' \), thus inconsistencies between \( I \) and \( I' \) can only come with shared symbols.

With this lemma and the first theorem we can show that minimal diagnoses can be computed with shared and mode literals only.

**Theorem 2** Let \( T^\forall \) be a description of an observed P2P system, \( F \) a consistent set of mode literals :

\[ \min_{\subset}(Diag(T)) = \min_{\subset}(\{(\otimes T^\forall_p)||sh,F_p)\} |\{F\}) \]

**Sketch of Proof** Let \( T \) be a the global description of an observed system s.t. \( T^\forall \equiv T \). By theorem 1 we know that \( \min_{\subset}(Diag(T)) = \min_{\subset}(T^\forall |\{F\}) \). We have \( T^\forall |\{F\} = T^\forall |\{sh,F_p\} |\{sh,F_p\} \) since the restriction of \( T^\forall \) on shared and faulty symbols do not delete any faulty symbols. Moreover, because \( T^\forall \equiv \otimes T^\forall_p \) we deduce by the lemma 2 that \( \min_{\subset}(Diag(T)) = \min_{\subset}(\{(\otimes T^\forall_p)|\{sh,F_p\}) |\{F\}) \).

### 3.2 Distributions with Trees

We now focus on the distribution of consistent diagnoses between diagnostics engines. Here we consider that any peer may be able to initiate a diagnosis and may ask its neighbor to help him for this. When receiving a request for a diagnosis by an initiator, a peer will also query its acquaintances, according to its observation values and will begin to answer to its initiator as soon as possible. Thus, the initial request will flood into the network top-down and answers will converge to the initial peer with a bottom-up traversal of the network. Implicitly, for a given request for a diagnosis, all peers will maintain who was its local initiator, and thus an implicit tree will be built in the network for each request.

We use this tree to efficiently compute the distribution of peers theories. Let us denote by \( A_p \) the subtree rooted in \( p \) and \( child(A_p,A_{p'}) \) a relation between \( A_{p'} \). A \( s.t. \) \( A_{p'} \) is a subtree of \( A_p \). We note by \( Sh_{A_p} \) the variables shared by \( A_p \) and any other peer in the distributed system. We note \( T^A_p \) the theory defined as the conjunction of all peers occurring in the the subtree rooted in \( p \).

\[
T^A_p = \begin{cases} 
T^\forall_p [|\{f_p,sh_p\}], & \text{if } \exists p \text{'s} \text{ child}(A_p,A_{p'}) \text{ is set.} \\
\{T^\forall_p \otimes \{sh_{p'}\}|\{sh_{A_p},f_{A_p}\}\} & \text{otherwise}
\end{cases}
\]

The next theorem shows that we can compute global diagnoses by gradually forgetting shared acquaintance which correspond to local acquaintance of a subtree.

**Theorem 3** Let \( T \) be the global description of an observed system, \( \text{Child}(A_p,A_{p'}) \) a relation defining a Tree on \( T \) rooted in \( r \) then :

\[ \min_{\subset}(Diag(T)) = \min_{\subset}(T^A_p) \]

**Sketch of Proof** We use the theorem 2 and inductively prove that \( \forall p, T^A_p = (\otimes T^\forall_p |\{sh_{A_p},f_{A_p}\}) |\{sh_{A_p},f_{A_p}\} \). Concerning the root \( r \), we note \( Sh_A=\emptyset \), consequently \( \min_{\subset}(T^A_p) \) only contains the set of minimal diagnosis.

Thus, intuitively, as soon as we know that a given variable cannot imply any inconsistencies in other parts of the tree, we remove it. As answers will go back to the root, peers will filter out useless variables, and, hopefully, will reduce the number and the size of possible answers.

### 4 Algorithm

In this section, we present our message-passing algorithm M2DT, standing for “Minimal Diagnoses by Distributed Trees”, algorithm 1. We call neighbor of \( p \) a peer that shares variables with \( p \). As previously, \( A \) stands for the distributed cover tree, dynamically built by the algorithm. We write \( A_p \) the subtree of \( A \) rooted in \( p \). For a tree \( A \) and a peer \( p, p \text{'s parents and p \text{'s children} will be included, by construction, in p \text{'s neighborhood}. Let us recall that \( T^A_p \) is defined as the theory of the observed subsystem defined by the conjunction of all peers occurring in the whole subtree \( A_p \). We call \( r \text{-impliant of } T^A_p \) a restriction of one implicant of \( T^A_p \) on its mode variables and shared vocabulary.

#### 4.1 A bird’s eye on M2DT

At the beginning, a given peer, called \textit{starter}, broadcasts a request of diagnosis (\texttt{reqDiag}) to its neighborhood. When a peer receives its first \texttt{reqDiag}, it sets the sender as his parent and broadcasts the request to its remaining neighbors, in order to flood the network. This first stage of the algorithm aims at building a distributed cover tree : as the request goes along the network, the relationship (\texttt{parent}, \( p \)) is set and defines the distributed cover tree \( A \). As soon as one peer knows that it is a leaf in \( A \), it answers by sending its r-implicants (\texttt{respDiag}) to its parent and thus begins the second stage of the algorithm.

When an intermediate nodes receives r-implicants from its children, there are two cases. If it already knows the role of all its neighborhood (parent, direct children and peers that can either occur deeper in the current subtree or elsewhere in the cover tree), it extends all new r-implicants by distributing it over its own r-implicants and those received from all other
children. It then filters out useless variables and send all resulting implicants to its parent. If it doesn’t know the role of all its neighborhood, it stores the received r-implicant for a future use. With this algorithm, global diagnoses converge to the starter peer. When a peer has received all termination messages from all its children, its sends its termination message to its parent(third and last stage of the algorithm). When the starter peer receives the termination message, we are sure that it already received the set of minimal diagnoses from all its children.

4.2 Structures and algorithm

A message can be a request of diagnosis reqDiag, a response respDiag or a notification of termination endDiag. The structure of a message msg is the following one:

\[
\text{msg.Type : takes its values in \{reqDiag, respDiag, endDiag\}, matching the three stages of the algorithm.}
\]

\[
\text{msg.Desc : defined only when msg.Type = respDiag, represents the descendants of the sender of the message that participated in building the considered implicant.}
\]

\[
\text{msg.rImpl : defined when msg.Type = respDiag is a r-implicant of the subtree rooted in the sender of the message.}
\]

A peer \( p \) sets its parent to the first peer in its neighborhood that sent it a reqDiag message. For all other reqDiag messages that \( p \) may receive, it adds the sender to the set NotChild. This set stores all peers that are not direct children of \( p \) (peers that can occur deeper in the subtree rooted in \( p \) that do not occur in this subtree). All peers \( p' \) that send to \( p \) at least one respDiag message are stored in Child. The array TChild, defined in each peer \( p \) only for its direct children, associates a peer \( p' \) with a DNF theory \( TChild[p'] \). \( TChild[p'] \) stores all r-implicants received so far from \( p' \). This set will be known to be complete when \( p \) will receive an endDiag message from \( p' \). In order to detect additional useless variables, \( p \) also stores in Desc all known descendant of \( p \) (peers occurring in the subtree rooted in \( p \)). All local variables of all peers are already deleted by the algorithm, but one may now consider as “local” a variable that is guaranteed to occur only in the current subtree and not elsewhere. This is the case for shared variables that are shared only by peers that occur in the current subtree.

To detect and notify termination, \( p \) maintains a list of peers from which messages are still waited. This list is called waitEnd and initially sets to all neighbors (Neighborhood). A peer leaves the list if it is the father, if it is not a direct descendant or if it is a direct child that notified termination.

4.3 Primitives of M2DT

checkEnd (waitEnd, \( p' \)) Checks and propagates the termination. First, it removes \( p' \) from the waitEnd list of \( p \). If waitEnd is empty, it sends the termination message and terminates.

extends \( (I, T^p_\chi, TChild, Desc) \) Extends the implicant \( I \) from \( p' \) by distributing it on the local theory \( T^p_\chi \) and all set \( TChild[p'] \) that are defined and different from \( p' \).

Algorithm 1 Peer \( p \) receives a message msg from peer \( p' \)

```
1: switch msg.Type
2: 
3: case : reqDiag /* and p ≠ starter*/
4: /* A distributed tree is built*/
5: if parent is not set then /* Flooding alg.*/
6: parent ← p'
7: send to all \( p \) neighborhood \( \setminus p' : \) msg [reqDiag]
8: else /* p' is not a direct child */
9: NotChild ← NotChild ∪ p
10: end if
11: /* Flush all stored implicants when the subtree is known */
12: if \{parent\} ∪ Child ∪ NotChild = Neighborhood
13: II ← flush(\( T^p_\chi, TChild, Desc \))
14: for all \( l \) ∈ II
15: send to parent msg [respDiag, I, Desc ∪ p]
16: end for
17: end if
18: /* p' is either our father or not a direct child */
19: checkEnd(waitEnd, \( p' \))
20: 
21: case : respDiag
22: /* Stores the diag, or extends and propagates it */
23: Child ← Child ∪ \( p' \)
24: Desc ← Desc ∪ msg.Desc
25: TChild[p'] ← TChild[p'] ∪ msg.rImpl
26: /* Extend msg.rImpl only if the subtree is already known */
27: if \{parent\} ∪ Child ∪ NotChild = Neighborhood
28: II ← extends(msg.rImpl, \( T^p_\chi, TChild, Desc \))
29: for all \( l \) ∈ II
30: send to parent msg [respDiag, I, Desc ∪ p]
31: end for
32: if \( p \) is the starter
33: Tresult ← min \( \prec \) (Tresult ∪ II)
34: end if
35: end if
36: 
37: case : endDiag
38: /* Notify termination of this child, and propagate if needed */
39: checkEnd(waitEnd, \( p' \))
40: end switch
```

This primitive, which is only called when the local subtree is entirely known, computes

\[
\begin{align*}
(T^p_\chi \cap (F_p, Sh_p) \otimes I \otimes TChild[p'])|{(Sh_A, F_A)}
\end{align*}
\]

One may notice that \( Sh_A \) is not directly known. It is deduced from Desc: we associate each shared variable with the unique identifiers of all peers that share it. Thus, one may check if all peers that share a given shared variables are “local” (i.e. in Desc) to the subtree, only with the help of the set Desc.

flush \((T^p_\chi, TChild, Desc)\) Sends the distribution of all implicants stored in the TChild arrays and the local theory. This primitive is called only when the local subtree is known to be complete for the first time with a reqDiag message, which means that the last unknown neighbor sent us a message “I’m not your direct child”.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
We thus have to flush all previously stored (if any) to our father. This primitive computes
\[(T_\lor \{p_{Sh}, F_{sh}\}) \otimes T_{Child}(p)]\]

One may notice that this primitive will be called for all leaves of the distributed tree \(A\).

### 4.4 Properties

In the following we assume a FIFO channel and no lost message. The acquaintance graph is logically connected and the global theory is satisfiable. Messages processing is considered as “atomic”, which simply means that messages are treated one by one by each peer.

Let us first emphasize some observations.

**Lemma 3** If a peer \(p\), sends a reqDiag to one of its neighbors, \(p'\), this message is the only one from \(p\) to \(p'\).

**Proof** A peer broadcasts the reqDiag to each of its neighbors (without the parent) just after having set as a parent the sender of the first received reqDiag. Because of the condition line 5, \(p\) does not have the opportunity to send other reqDiag. Concerning respDiag and endDiag they are sent to the parent only.

If we now focus on the first event carries out by each peer:

**Lemma 4** All peers, except starter, will receive a first event, which will be a reqDiag message.

**Proof** To receive a respDiag or an endDiag, \(p\) must have been a parent of some peer. But to become a parent, \(p\) must send a reqDiag to at least one peer, and thus \(p\) would have to receive reqDiag first. Consequently, since the acquaintance graph is connected, and because a peer broadcasts reqDiag to its neighbors, then each a peer will receive a first reqDiag (we rely on the well known flooding algorithm in a graph to ensure this).

With these lemmas we have the following property:

**Property 1 (Distributed cover Tree)** The relation (parent, p), built when \(p\) receives its first reqDiag, defines a distributed cover tree over the distributed system.

**Sketch of Proof** Let \(n\) be the number of peers, req\((p)\) be the reception of the first diagnosis request by \(p\). Since the peer starter does not get a parent, by the previous lemma we know that flooding reqDiag will build \(n-1\) connections (parent, p). Suppose a cycle is defined by these connections and an order \(< t\). req\((p) < req\((p')\) if req\((p)\) is former than req\((p')\). Let us take \(p\) in the cycle, there exists \(p'\) in the cycle s.t. \(p\) got \(p\) as parent then req\((p) < req\((p')\).

If we follows the "parent" connection in the cycle, we have by transitivity that req\((p') < req\((p)\). Consequently, we cannot have cycle by the "parent" connection.

Now we know that a distributed cover tree will be built but, at this point, a peer \(p\) will only know its parent, but not its direct children. This can be deduced by the respDiag messages, when all children of \(p\) will send him their r-implicants. respDiag messages are only sent when the state of all neighbors of \(p\) is known (Parent, Child, NotChild).

**Lemma 5** Let \(p\) be a peer, \(p'\) be one of its neighbors. If \(p'\) does not get \(p\) as father, \(p\) will receive a reqDiag from \(p'\).

**Sketch of Proof** Let \(p'\) a neighbor of \(p\) that does not accept \(p\) as its father. When \(p\) sent to it a reqDiag, \(p'\) already had a parent in order to refuse \(p\) as its parent. Consequently \(p\) had also sent to \(p'\) a reqDiag.

With this lemma, we can easily show the main property:

**Property 2** Let \(A_p\) be the subtree rooted in \(p\) and built by the algorithm, \(T_{A_p}\), the theory of \(A_p\) as previously defined, \(p\) will send to its parent the r-implicants of \(T_{A_p}\).

**Sketch of Proof** Let us recursively show this property with the maximal depth, \(d_{max}\), of the subtree \(A_p\), rooted in \(p\). If \(d_{max} = 0\), \(p\) is a leaf, none of its neighbors gets it as parent. Nevertheless, with the previous lemma, we know that each of them will send a reqDiag to it. Consequently \(p\) will know the state of its neighbors and satisfy the condition (Father \(\cup\) Child \(\cup\) NotChild = Neighborhood). Then \(p\) will send all r-implicants of \(T_{A_p}'\) computing by the primitives flush. If \(d_{max} \geq 0\), suppose the property true for all children \(p'\) of \(p\); we thus guarantee that \(p\) will receive from each of them their r-implicants and will store them in its T Child arrays. Concerning the other neighbors (NotChild), \(p\) will receive from them a reqDiag. Consequently, \(p\) will know the state of its neighborhood and satisfy the condition (Father \(\cup\) Child \(\cup\) NotChild = Neighborhood). Since the global theory is satisfiable \(p\) will be able to build at least one r-implicant of \(T_{A_p}'\) either by the method extends or by the method flush.

Since \(T_{Result}\) is minimized, the peer starter will save in it the minimal diagnoses. The correction, the completeness and the termination of the algorithm is a direct consequence of the previous property. The termination is shown by this property:

**Property 3** The last event carried out by a peer is sending the endDiag message to its parent.

**Sketch of Proof** Similarly to the previous theorem, we can recursively show this property by the maximal depth, \(d_{max}\). If \(d_{max} = 0\), \(p\) will receive a reqDiag from all its neighbors and the set waitEnd will be empty. Then \(p\) will send the message end after the set of r-implicants of \(T_{A_p}'\). If \(d_{max} \geq 0\), since \(p\) will receive reqDiag from peers in NotChild and endDiag from its children. Then, the set waitEnd will be empty. As soon as possible, it will send r-implicants to its parents, \(p\) will send endDiag as its last message.

---

**Fig. 2 – M2DT algorithm**

**Example 5 (M2DT Illustration)** A customer has made a hire purchase by internet whereas his credit card was not valid : the service CCS observed ¬valCC and the service ES observed hPurch. OVS starts the analysis and sends a diagnosis request to ES. ES begins the computation of \(T_{ES}(s_{hs}, F_{ES})\) and forwards...
the request to its neighbors HPS and LAS. HPS and LAS receive the diagnosis request from ES and both forward a diagnosis request to CCS. CCS receives the request from HPS then forwards it to LAS. When LAS receives request from CCS it has already received one from ES, so it does not answer. At this step, LAS has received a message from all its neighborhood, then it starts to send its implicants to ES. When CCS receives the request from LAS, it does not answer and sends its implicant ¬valCC to HPS. Simultaneously, ES gets ¬laAprvl from LAS and HPS gets ¬valCC from CCS. At this step, ES did not receive any message from any of its neighbors, unlike HPS, which can send to ES the implicant ¬bkAprvl ∧ ¬valCC built from the 4th implicant of its theory and ¬valCC. At this step, ES received a message from its neighbors. It buieses. Many other works try to take advantage of the system topology, for instance by using decomposition properties of the model [Darwiche, 1998; Provan, 2002]. Similarly [Kurien et al., 2002] search a diagnosis into a partitioned graph by assigning values for shared variables and maintaining a local consistency. The global diagnosis is thus distributed in all local diagnoses. This method however do not guarantee the minimality and suppose a global system description, which is not our case. Some work [Biteus et al., 2006] synchronizes local diagnosis from each agent (peer) such that it will contain a compact representation of global diagnoses at the end. The algorithm search for the set of minimal diagnosis with minimal cardinality whereas we look for the set of all minimal diagnosis. In [Roos et al., 2003], agents update their sets of local diagnoses in order to be consistent with a global one. However, the algorithm cannot guarantee that any combination of agent local minimal diagnoses are also a global minimal diagnosis.

5 Related Works

Our approach has been preceded by many pieces of work. Methods from [Masson and Johnson, 1989] and [Beckstein et al., 1993] extend the ATMS principles of [de Kleer, 1986] in order to incrementally computes the set of conflicts in a distributed framework. However, those methods have still to wait for all conflicts before having a chance to get the first diagnosis. Many other works try to take advantage of the system topology, for instance by using decomposition properties of the model [Darwiche, 1998; Provan, 2002]. Similarly [Kurien et al., 2002] search a diagnosis into a partitioned graph by assigning values for shared variables and maintaining a local consistency. The global diagnosis is thus distributed in all local diagnoses. This method however do not guarantee the minimality and suppose a global system description, which is not our case. Some work [Biteus et al., 2006] synchronizes local diagnosis from each agent (peer) such that it will contain a compact representation of global diagnoses at the end. The algorithm search for the set of minimal diagnosis with minimal cardinality whereas we look for the set of all minimal diagnosis. In [Roos et al., 2003], agents update their sets of local diagnoses in order to be consistent with a global one. However, the algorithm cannot guarantee that any combination of agent local minimal diagnoses are also a global minimal diagnosis.

6 Conclusion

We proposed a distributed algorithm to compute minimal diagnoses of a distributed setting in anytime, with the help of a distributed cover tree of the acquaintance graph of peers. Our algorithm takes advantage of the DNF representation of local theories of peers in order to compute global diagnoses without conflicts. However, one has to notice that in practice, peers don’t have to rewrite their local theories in DNF. They may compute answers to requests “on the fly” and thus allow our algorithm to work on CNF encoding of peers. Our approach have many other advantages: we never compute the set of conflicts before computing the diagnoses, we take advantage of the natural structure of the network, which can be decomposed by the small number of shared variables, we take advantage of the distributed cpu power of the whole network and, and last, we restrict the vocabulary of diagnoses as soon as possible.

Références


Towards Active Diagnosis of Hybrid Systems

Mehdi Bayoudh1 and Louise Travé-Massuyès1 and Xavier Olive2

1LAAS-CNRS, Université de Toulouse, France
2Thales Alenia Space, FRANCE
{bayoudh, louise}@laas.fr, xavier.olive@thalesaleniaspace.com

Abstract

In this paper we propose a modeling framework aimed at hybrid systems diagnosing. In this framework, the behavior of the hybrid system is modeled by a hybrid automaton that describes system transitions between operating nominal and faulty modes. After a fault occurrence, the system behavior is modeled by an anticipated fault mode with associated continuous dynamics. Control actions in fault modes may be used to perform active diagnosis. The proposed diagnosis framework allows to abstract the continuous dynamics changes in terms of discrete events. A hybrid language that contains natural discrete events and events capturing continuous dynamics is hence defined. Based on this language, the diagnoser approach is used and allows to perform on-line diagnosis. The active diagnosis problem is formulated as a conditional planning problem. From an ambiguous state of the diagnoser the plan defines how to find a controllable paths leading to a non ambiguous states. Search of active diagnosis actions is guided by the observable response of the system.

1 Introduction

On-line diagnosis is often approached as a passive task that takes as input the available observations provided by the sensing devices instrumenting a physical system and returns an estimation of its state, often interpreted in terms of the status of each of the components. However diagnosis is originally defined as a process [Hamscher et al., 1992] that interlinks the determination of a belief state and the proposal of new tests that provide additional information allowing the diagnoser to refine the belief state and ultimately end with a non ambiguous state estimation. This way to go is quite common for solving post-mortem diagnosis problems, and the diagnosis is often formulated as a test sequencing problem or related in some way to testing [Struss, 1994] [Abramovici et al., 1999] [Nicolaides and Zorian, 1998]. The proposed tests can take the following forms or a mixture of them:

- new variable to be sensed
- new input pattern defined by specific values or signals to be applied to the system
- configuration in which the system should be put
- time constraints for achieving the operating tasks are not severe and one can consider to use momentarily the inputs for diagnosis purposes
- embedded electronic controllers acting on physical systems impose discrete switches that result in numerous configurations (or operating modes) that are as much possibilities for active diagnosis.

On the other hand, active diagnosis is dictated by reliability and availability requirements. Reconfiguration actions can indeed be dangerous if the belief state is too ambiguous.

This paper presents preliminary ideas to achieve active diagnosis in a hybrid system framework. Starting with an ambiguous belief state, our method calls for diagnosability analysis results to determine a new system configuration in which fault candidates can be discriminated. The command inputs to be applied to the system to drive it into this configuration are then determined, paying attention to avoid states that could be dangerous for the system.

Existing works are first surveyed in section 2 in the framework of discrete event systems (DES) and continuous systems (CS). Our hybrid modeling framework is then presented in section 3, followed by the background results that are used by the active diagnosis approach in section 4. The active diagnosis problem and the proposed active diagnosis scheme are detailed in section 5. Illustrative example is presented in section 6. Finally section 7 concludes the paper.
2 Problem Assessment and Related Work

There are very few works dealing with active diagnosis understood as active excitation of the system through its inputs to achieve diagnosis. [Sampath et al., 1998] can be mentioned as one of the only works proposing an approach for active diagnosis of Discrete Event Systems (DES). A DES is modeled by a finite state machine, and active diagnosis is formulated as a supervisory control problem [Ramadge andWonham, 1989]. The novelty of the paper is to devise the controller so that specific actions that may drive the system into non diagnosable regions are forbidden. The system is hence "actively" diagnosable, allowing non ambiguous diagnosis to be performed.

In the field of continuous systems (CS), [Niemann, 2006] and [Niemann, 2005] are certainly the most representative works. An approach for Active Fault Diagnosis (AFD) of parametric faults is proposed for closed loop continuous systems. Auxiliary signals are introduced and a fault signature matrix in connection with parametric faults is defined. This fault signature matrix is used for fault detection and isolation. When diagnosis based on the structure of the fault signature matrix is not possible, active diagnosis is performed thanks to the auxiliary inputs. Auxiliary input signals are designed so that the effect on system performance is minimized, but it becomes possible to detect/isolate parametric faults in the system. Through the above examples, it comes that active diagnosis must be seen as an integrated control and diagnosis problem. The diagnosis objectives must be superposed to the normal operation control objectives. The system’s operation is required, in the worst case, to remain safe and, in the best case, to preserve normal performances when specific inputs are applied to drive the system into state space regions that exhibit the appropriate symptoms.

Our objective is to achieve active diagnosis for systems that show continuous and discrete dynamics, namely hybrid systems. In this framework, both continuous and discrete control actions can be used, in an interlinked way, to perform active diagnosis. These interlinked actions ultimately act by putting the system in a goal configuration, i.e. a goal behavioral mode. It can be added to model all the non anticipated faulty situations.

- Σ is the set of events. Events correspond to discrete control inputs, spontaneous mode changes and fault events. Events corresponding to spontaneous mode changes are triggered upon guards that depend on continuous variables.
- Σo ⊆ Σ is the set of observable events.
- Σuo ⊆ Σ is the set of non observable events. Without loss of generality, we assume that fault events are unobservable (otherwise, these faults are obviously diagnosable).
- Σ = Σuo ∪ Σo
- T is the transition function, T : Q × Σ → Q.
- C is the set of system constraints linking continuous variables. It represents the set of differential and algebraic equations modeling the continuous behavior of the system.
- (ζ0, q0) ∈ ζ × Q, is the initial condition of the hybrid system.

The discrete part of the hybrid automaton, given by M = (Q, Σ, T, q0), is a discrete automaton that describes the discrete dynamics of the system, i.e. the possible evolutions between behavioral modes of M.

The continuous behavior of the hybrid system is modeled by an underlying continuous system $Ξ = (ζ, Q, C, ζ_0)$ that describes the whole continuous behavior of the system. Notice that in $Ξ$ transitions between modes are implicit and consequently not constrained in any way. We hence call this system a multimode system.

The underlying continuous behavior in each mode $q_i$ is modeled by a set of constraints $C_i$.

The hybrid behavior is the result of the contribution of the underlying continuous system $Ξ$ and the underlying discrete event system $M$.

3 Hybrid Framework Modeling

A hybrid system is modeled as a hybrid automaton [Henzinger, 1996], $S = (ζ, Q, Σ, T, C, (q_0, ζ_0))$, where:

- ζ is the set of continuous variables, which includes 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 $q_i ∈ Q$ represents a behavioral mode of the system. It includes nominal and anticipated fault modes. An unknown mode

---

1We assume that the set of system observable variables is unique in all system modes. This assumption is generally verified when the set of system’s sensors is permanent, and do not depend on the system mode.
mode \( q_i \). The \( q_i \)-mirror signature of mode \( q_j \) is given by the vector \( S_{j/i} = [s_{1j/i}, s_{2j/i}, ..., s_{Nr(q_i)j/i}]^T \), where \( \zeta_{OBS_{q_j}} \) denotes the value of observable variables when the system mode is \( q_j \).

The signature of a mode \( q_i \) is the vector obtained by the concatenation of all the mirror signatures of \( q_i \), \( \text{Sig}(q_i) = [S_{1i}^T, S_{2i}^T, ..., S_{Nr(q_i)i}^T]^T \), where \( m \) is the number of system modes.

### Abstraction of the continuous dynamics in terms of discrete events

Let us assume that the dynamics of the discrete control inputs are slower than the dynamics of residual generators (mode signatures establish between two consecutive discrete events).

The function \( f_{CS,DES} \) associates an event issued from the continuous domain that represents the change of the mode signature, to each discrete transition of the underlying DES.

This function aims to define \( \Sigma_{Sig} \), as the set of discrete events issued from the abstraction of continuous dynamics of the multimode system.

\[
f_{CS,DES}: Q \times T(Q, \Sigma) \rightarrow \Sigma_{Sig}
\]

\[
(q_i, q_j) \rightarrow \begin{cases} \Sigma_{Sig} & \text{if Sig}(q_i) \neq \text{Sig}(q_j) \\ \Sigma_{Sig} & \text{if Sig}(q_i) = \text{Sig}(q_j) \end{cases}
\]

- \( \Sigma_{Sig} \) is a set of observable events, generated when the mode signature of the source mode is different from the mode signature of the destination mode.
- \( \Sigma_{NoSig} \) is a set of unobservable events generated when the mode signature of the source mode is equal to the mode signature of the destination mode.
- \( \Sigma_{Sig} \) is defined as \( \Sigma_{Sig} \cup \Sigma_{NoSig} \).

### 4.2 Hybrid Language and Hybrid Trajectories

The abstraction of the continuous dynamics changes in terms of discrete events allows us to define the language of the hybrid system, which describes the evolution of the system behavior. We denote by \( \Sigma_{hybrid} = \Sigma \cup \Sigma_{Sig} \) the alphabet that contains "natural" discrete events and events modeling signature switches.

We model the behavior of the hybrid system as a prefix closed language \( L(S) \subset \Sigma_{hybrid} \) over the event alphabet \( \Sigma_{hybrid} \), where \( \Sigma_{hybrid}^* \) denotes the set of all finite strings of elements of the set \( \Sigma_{hybrid} \) including the empty string (\( \Sigma_{hybrid}^* \) is called the Kleene Closure of \( \Sigma_{hybrid} \) [Ramadge and Wonham, 1989]). A trajectory of the hybrid system is represented by a string of events of the hybrid alphabet \( \Sigma_{hybrid}^* \).

The hybrid language \( L(S) \) can be generated by its finite state generator representation [Ramadge and Wonham, 1989]. In this paper, this automaton is called the behavior automaton (denoted \( B_A(S) \)) and mixes both "natural" discrete events and signature switches.

---

In our approach, nominal and fault modes have the same status and the signature of a given mode anticipates how it should be seen in terms of the indicator tuples of the different modes of the system (including itself).

### 4.3 Hybrid system diagnosability

This paper is based on diagnosability definition for hybrid systems proposed in [Bayoudh et al., 2008a] given as follows:

#### Definition 2

A fault event \( f \) is diagnosable if its occurrence can always be detected after a finite set of continuous and discrete observations i.e. after a finite sequence of observable events and a finite set of continuous variable observations. The system is said to be diagnosable if and only if all the anticipated faults are diagnosable.

This definition provides the following result in the hybrid language framework:

**Proposition 1** The hybrid system is diagnosable if \( \forall f_i, \exists n_i \in \mathbb{N} \) such as: \( \forall s_F, t \in L(S) \), such that \( s_F \) ends with the occurrence of \( f_i \), and \( t \in L(S) \) is a continuation of \( s_F \), \( ||t|| \geq n_i \Rightarrow (\forall w \in L(S) : P_{\Sigma_{hybrid_0}}(w) = P_{\Sigma_{hybrid_1}}(s_F; t) \Rightarrow f_i \in w) \), where \( P_{\Sigma_{hybrid_0}} \) is the projection operator on the set of observable events of \( \Sigma_{hybrid} \).

Bayoudh et al., 2008a] provided the following result that relies on the diagnoser approach of Sampath et al., 1995. The diagnoser \( Diag(B_A(S)) = (Q_{Diag}, \Sigma_{Diag}, T_{Diag}, \eta_{Diag}) \) is a deterministic finite state machine built from the behavior automaton \( B_A(S) \).

- \( \eta_{Diag} = \{(q_0, \emptyset)\} \) is the initial state of the diagnoser.
- \( \Sigma_{Diag} = \Sigma_0 \) is the set of observable events of the system.
- \( Q_{Diag} \) is the set of states of the diagnoser: \( Q_{Diag} \subseteq 2^Q \times 2^T \) or \( Q_{Diag} \subseteq P(Q \times P(\Sigma_F)) \), where \( P(E) \) 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. For example, when the state of the diagnoser of figure 6 is \( (\{1F_1, \{f_1\}\}, \{1F_2, \{f_2\}\}, \{1F_3, \{f_3\}\}) \), it means that the system is in the state 1F1 after the occurrence of \( f_1 \) or in 1F2 after the occurrence of \( f_2 \) or in 1F3 after the occurrence of \( f_3 \).
- \( T_{Diag} \) 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.

**Proposition 2** The hybrid system \( S = (\zeta, Q, \Sigma, T, C, (\zeta_0, q_0)) \) is not diagnosable if

- the associated diagnoser computed from the corresponding behavior automaton contains an uncertain cycle, i.e. a cycle in which there is at least one \( F_i \)-uncertain diagnoser state for some \( F_i \).
- the states of the behavior automaton involved in the different diagnoser cycling states also define a cycle in the behavior automaton.
5 Active Diagnosis of Hybrid Systems

Let us assume that a hybrid system as described in section 3 is continuously monitored and that its state is tracked following, for instance, the approach proposed in [Bayoudh et al., 2008b]. Assume that the current belief state returned by the diagnoser is faulty and ambiguous, i.e. several faults are candidate. This is the starting point of an active diagnosis session.

In contrast to [Sampath et al., 1998] that forbids non diagnosable regions with appropriate control actions, our approach is based on driving the system towards diagnosable regions. These regions correspond to non ambiguous states of the hybrid system diagnoser.

What is important to notice is that even when the conditions for non diagnosability as stated by proposition 2 hold, there may be a way to enforce a sequence of transitions to drive the system towards a non ambiguous state of the diagnoser. Indeed, an uncertain cycle of the diagnoser (and the corresponding behavior automaton) only indicates that the system may get stuck in the cycle. It is hence a worst case analysis. The conditions for active diagnosis are weaker.

5.1 Controllable and induced controllable events

Active diagnosis requires to define the notion of controllable events. Let us call $\Sigma_c \subseteq \Sigma$ the set of controllable events.

**Definition 3 (Controllable event)** Controllable events fall in one of the categories below:

- discrete control events
- events corresponding to spontaneous mode changes when the departure mode continuous dynamic model is controllable in the sense of [Terrell, 1999]. This means that there always exists a continuous control law that leads to the occurrence of such event.

The set of possible transitions outgoing fault modes represents all the control actions that can be done to perform active diagnosis. The set of allowed control actions is different for the different fault modes, and is a mean to account for safety constraints.

**Definition 4 (Induced controllable event)** Events whose occurrence always follow the occurrence of a controllable event are called induced controllable events and form the set $\Sigma_{ic}$.

Induced controllable events model the observable system response after a control action (applying a discrete input event or a continuous input signal). Induced controllable events are a subset of $\Sigma^{Sig}$. Indeed, $R_{ij} \in \Sigma^{Sig}$ are associated to mode signature changes. $R_{ij}$ is an induced controllable event, noted $R_{ij}^c$, if the mode change is controlled by a controllable event.

Controllable events are those that provide means to act on the system. Induced controllable events are those that manifest the reaction of the system and allows us to discriminate ambiguous situations. Given $\Sigma_{uc} \subseteq \Sigma$ the set of non controllable events, we have $\Sigma = \Sigma_{uc} \cup \Sigma_c \cup \Sigma_{ic}$.

**Definition 5 (Controllable path)** Consider the hybrid system behavior automaton and its associated hybrid language $\Sigma_{hybrid}$ as defined in 4.2, a controllable path is a string of controllable and induced controllable events.

A controllable path in the behavior automaton corresponds to a controllable observable path in the corresponding diagnoser.

5.2 Active diagnoser

Our idea is to use the diagnoser to guide the search for the sequence of actions that will desambiguate a belief state. However in order to suit active diagnosis purposes, the diagnoser must be modified into an Active Diagnoser. Indeed the control actions that appear in the diagnoser are supposed to be observed but not applied. In particular, a control event associated to a transition outgoing an ambiguous state only is observed in at least one of the underlying faulty states. In our case, we want to actively apply the control event, which means that it must be applicable in all the underlying faulty states, otherwise it means that the control is forbidden and may be dangerous in some underlying situations.

The diagnoser is hence modified accordingly. Given an ambiguous state of the diagnoser, outgoing transitions associated with controllable events are removed if there is no corresponding transition outgoing from all the corresponding states of the behavior automaton.

5.3 Search of an active diagnosis plan

Active diagnosis consists of exciting the hybrid system to exhibit additional observations. Given an uncertain state of the active diagnoser, the active diagnosis problem is how to find a controllable path leading to a certain state. From the uncertain state the active diagnosis is performed by applying a control action, observing the system reaction, and deciding about the next control action. This action depends on the last observed induced controllable event. This problem can be formulated as a conditional planning problem [Bertoli et al., 2001] [Jimenez and Torras, 2000]. The active diagnoser can be seen as a AND-OR graph, and a MINIMAX algorithm can be applied to resolve a conditional planning problem.

![Figure 1: The active diagnosis seen as a planning problem](image)

- "OR" nodes correspond to control actions

Alban Graußen, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
6 Illustrative Example

Consider a hybrid system consisting of a three tanks of water, $T1, T2$ and $T3$. Valves $V1$ and $V2$ allow the flow transfer between tanks. Valves are controlled by discrete control inputs $openV_1, openV_2, closeV_1$, and $closeV_2$. The system is equipped with two level sensors that measure the level of water in each tank. Hence, water levels $h_1, h_2$ and $h_3$ are observable. The behavior of the system in nominal modes (no fault) is described in figure 3. Every nominal mode in figure 3 models a configuration of the system as shown in table 1.

<table>
<thead>
<tr>
<th>nominal mode</th>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>opened</td>
<td>opened</td>
</tr>
<tr>
<td>N2</td>
<td>closed</td>
<td>opened</td>
</tr>
<tr>
<td>N3</td>
<td>closed</td>
<td>closed</td>
</tr>
<tr>
<td>N4</td>
<td>opened</td>
<td>closed</td>
</tr>
</tbody>
</table>

Table 1: The system configuration in nominal modes

The observable continuous behavior in every mode (nominal or faulty) is described by constraints linking observable variables given in table 2. Boolean consistency indicators are associated to every constraint and allow one to check the consistency between observations and system model (see table 3).

\[
\frac{dh_3}{dt} = 0 \iff r_4 = 0 \quad \frac{dh_3}{dt} < 0 \iff r_5 = 0 \\
\frac{dh_3}{dt} = 0 \iff r_6 = 0 
\]

Table 2: Set of continuous constraints in each operating mode

Table 3: The consistency indicators

considered only once in the mode signatures of the system. Given $[r_1, r_2, r_3, r_4, r_5, r_6]$, the vector of all system residuals, mode signatures are obtained by evaluating this vector using system observations in every mode. The mode signatures of the system are given in table 4.

Abstraction of the continuous behavior in terms of discrete events:

Let’s consider the case when the fault event ($f_1, f_2$ or $f_3$) occurs in the nominal mode $N1$. The corresponding behavior automaton is shown in figure 5. Events $R_{c1}, R_{c2}, R_{c3}, R_{c4}$ and $R_{c6}$ ($f_1, f_2$ or $f_3$) correspond to the observable switches of mode signatures that follow control inputs. They belong to the set of induced controllable events $\Sigma_{ic}$. $R_{c6}$ corresponds to the observable switch of mode signature after the occurrence of a fault event ($f_1, f_2, f_3$ or $f_4$). For sake of clarity, non observable events of $\Sigma_{non}$ are not represented.

The diagnoser of the three-tank system is computed from the behavior automaton. Let focus on the part of the diagnoser shown in figure 6. The occurrence of the fault event $f_1, f_2$ or $f_3$ is detected by the observation of the
Towards Active Diagnosis of Hybrid Systems

Table 4: Mode Signatures of the three tank system

<table>
<thead>
<tr>
<th>Departure mode</th>
<th>Arrival mode</th>
<th>Associated event</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>{1F1, 1F2, 1F3}</td>
<td>Rof1</td>
</tr>
<tr>
<td>1F1</td>
<td>2F1</td>
<td>Rof1</td>
</tr>
<tr>
<td>{1F2, 1F3}</td>
<td>{2F2, 2F3}</td>
<td>Rof2</td>
</tr>
<tr>
<td>2F2</td>
<td>3F2</td>
<td>Rof2</td>
</tr>
<tr>
<td>2F3</td>
<td>3F3</td>
<td>Rof2</td>
</tr>
<tr>
<td>1F1</td>
<td>1F1</td>
<td>Rof4</td>
</tr>
<tr>
<td>{2F2, 2F3}</td>
<td>{1F2, 1F3}</td>
<td>Rof4</td>
</tr>
<tr>
<td>3F2</td>
<td>2F3</td>
<td>Rof4</td>
</tr>
<tr>
<td>3F3</td>
<td>2F3</td>
<td>Rof4</td>
</tr>
</tbody>
</table>

Figure 5: A part of the behavior automaton of the three-tank system

Table 5: observable events associated to mode signature changes

<table>
<thead>
<tr>
<th>Departure mode</th>
<th>Arrival mode</th>
<th>Associated event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sig(N1) = Sig(N2) = Sig(N3) = Sig(N4) = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sig(1F1) = Sig(1F2) = Sig(1F3) = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sig(2F1) = Sig(3F1) = Sig(2F3) = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sig(3F3) = Sig(4F3) = 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: The active diagnoser of the three tank system

The presence of the uncertain cycle [[{2F2, {f2}}, {2F3, {f3}}], {{2F2, {f2}}, {2F3, {f3}}}] (cycle defined by red transitions in figure 6) proves (proposition 2) that the language of the hybrid system is not diagnosable.

The non diagnosability of the system language is due to the non diagnosability of faults f2 and f3 pointed out by the uncertain cycle shown in figure 6. However, we show that performing active diagnosis allows us to diagnose the system with certainty. The active diagnosis consists of searching a conditional plan that permits to leave the uncertain state of the diagnoser and reach a certain state. Consider the active diagnoser. The active diagnosis define the set of plans to leave uncertain states. These uncertain state may be crossed by uncertain cycle or not ({{2F2, {f2}}, {2F3, {f3}}}). In our approach, the active diagnoser is seen as a AND-OR graph with control inputs in AND nodes and induced controllable events in the OR nodes. From the uncertain state of the diagnoser the active diagnosis plan is given by the MINIMAX algorithm. An active diagnosis plan defines a set of controllable paths from the uncertain state of the diagnoser to a certain state. Given the system diag-
noser, the occurrence of a fault event \( f_1, f_2 \) or \( f_3 \) is detected by the observable events \( R_{\omega_1} \) and puts the diagnoser in the uncertain state \( \{ (1F_1, \{ f_1 \}), (1F_2, \{ f_2 \}), (1F_3, \{ f_3 \}) \} \). From this uncertain state the active diagnosis plan is: \( close_{V_1}, \) if \( R_{\omega_2} \) \( close_{V_2} \) Else [ ]

7 Conclusion
This paper deals with the problem of diagnosing hybrid systems that exhibit continuous and discrete event dynamics. The abstraction of the continuous dynamics in terms of discrete events allows one to use discrete event techniques to perform diagnosis. Based on these results, the diagnoser approach is used to perform on-line diagnosis. When the diagnoser is blocked in an ambiguous state, the active diagnosis process is needed. Concepts of controllable path, controllable induced events and active diagnoser are introduced and allows us to formulate the active diagnosis as a conditional planning problem. From an ambiguous state, the active diagnosis consists of defining a controllable path leading to certain a state. The choice of a control action depends on the observed response of the system after the previous action. Several problems remain. In particular, it is not clear when it is better to wait for more events to occur that may discriminated an ambiguous situation or to start an active diagnosis session. Finally, given an ambiguous diagnosis, conditions of the existence of the active diagnosis will be studied in future works, and diagnosability definition w.r.t. active diagnosis must be considered.

References


Model-Based Diagnosis of Hybrid Dynamical Networks for Fault Tolerant Control

Lachlan Blackhall ∗ and Priscilla Kan John †
The Australian National University, Canberra, ACT, 0200, Australia
lachlan.blackhall, priscilla.kanjohn@anu.edu.au

Abstract

A method for determining (diagnosing) the operational mode of a hybrid dynamical network using model based diagnosis is presented. This methodology unifies discrete diagnosis capabilities with the continuous time dynamics of hybrid dynamical networks. This method is novel in that the diagnosis is not performed on the output trajectory of the network but rather on the fundamental governing dynamics of the network. This novelty allows for the diagnosis of arbitrary switching events in the network. The chosen methodology can be partially decentralised and provides optimal diagnoses even when only a portion of the network can be observed. This offers performance and implementation benefits for large distributed networks. Computational results are presented for the case where the mode changes are purely structural. That is, where only the interconnection structure of the network is changing. The results indicate that this methodology has some benefits for the fault-tolerant control of hybrid dynamical networks.

1 Introduction

Effectively controlling dynamical networks (systems where there are very many constituent components physically linked together) poses many challenges. These challenges are somewhat more significant in the presence of faults within the network. Faults drastically alter the fundamental governing dynamics (how the constituent systems or components are linked together and affect each other) of the network rendering the original, fault-free control methodology severely degraded or useless.

We thus require that an effective network controller be able to detect that a fault has occurred and change control strategy accordingly. Choosing an appropriate control strategy for a given configuration is achievable and has been the focus of extensive work in the field of control theory [Khalil, 2002]. Additionally these control strategies can usually be computed off-line. What is lacking is a systematic on-line methodology to automate the process of detecting the occurrence of a fault(s) and choosing the most appropriate control strategy. Additionally we wish to minimise the fault monitoring required to still achieve the global performance objectives. Allowing the fault detection and recovery strategy to run in a distributed manner would be an additional benefit.

The discrete diagnosis community has undertaken considerable work in the area of fault diagnosis and we propose to apply these diagnosis methods to detect and recover, in real time, from faults experienced in hybrid dynamical networks. The unification of the discrete diagnosis capability and continuous time operation of dynamical networks is discussed in depth in Section 3.

Much of the previous work focuses on detecting faults by measuring deviations of a dynamical system (a subset of a hybrid dynamical network) from the ideal or nominal trajectory. In [Blanke, 2006] a residual generating transfer function is created that forces the residual to become non-zero in the presence of a fault. In [Narasimhan et al., 2000] a similar approach is used. Both works categorise faults through analysis of the system output, requiring that faults be manifest in the residual error between the system output and a nominal trajectory. In dynamical networks, where there are many interconnected systems, it is theoretically and practically challenging to determine a nominal trajectory for each system and the network as a whole. Our diagnosis methodology rather focusses on operational mode changes that can be determined by changes in the fundamental governing dynamics of the network.

In [McIlraith et al., 2000] the authors approach diagnosis as a model selection problem. An initial set of qualitative candidate diagnoses are conjectured and then refined using parameter estimation and model fitting techniques. The assumptions that controller actions are responsible for all events and the inability to deal with multiple sequential faults limits the applicability of this work to dynamic networks where these assumptions are not always valid.

Other work has focussed on the prediction and selection of the dynamical mode that generates the trajectory that best describes the output of the dynamical system. In [Hofbaur and Williams, 2002] the modeling framework proposed is a probabilistic hybrid automata (PHA). These merge hidden
Markov models (HMM) with continuous dynamical system models to achieve diagnosis. In [Lerner et al., 2002] systems are modelled as dynamic Bayesian networks (DBN). This representation is, in principle, able to deal with multiple simultaneous failures and is robust to parameter drift. In both approaches tracking a number of trajectories becomes intractable and they are required to use heuristics to reduce the computational burden.

Model-based diagnosis is concerned with determining the sequence of operational modes of a discrete system, where it is assumed that the occurrence of a fault could cause changes in the operational mode of the system. Model-based diagnosis assumes that the modes and events can be deterministically related via a model (e.g. an automata). Thus, by using observations of the system in question it is possible to make a diagnosis (determine the operational mode) that is consistent with the observations. Where multiple systems have common functionalities, one can make computationally related diagnoses using only local diagnosis calculations to ensure global consistency.

In this paper, we propose a diagnosis approach for hybrid dynamical networks with partial observations based on the detection of changes in the fundamental dynamics of the network. Our approach is both distributed and model-based, requiring only local diagnosis calculations to ensure global consistency.

2 Preliminaries

2.1 Graphs and Networks

Networks have emerged as a way of describing and analysing large numbers of individual elements or agents that interact. Understanding the interaction of these elements is of considerable interest in order to better understand the collective operation of the whole. We begin with a preliminary definition given in [Diestel, 2005].

**Definition 2.1** (Directed Graph). A directed graph $G = (V, E)$ is a pair $(V, E)$ of disjoint sets and two maps, $\text{init}(E) : E \rightarrow V$ and $\text{ter}(E) : E \rightarrow V$, that assign to each edge an initial and terminal vertex respectively, where $E \subseteq V \times V$. In this way a given edge is said to be directed from $\text{init}(E)$ to $\text{ter}(E)$. To avoid ambiguities we assume that $V \cap E = \emptyset$. We thus have that the elements of $V$ are vertices and the elements of $E$ are edges that represent a directed relationship between two vertices.

From these basic definitions we are now able to define a network.

**Definition 2.2** (Network). A network is the directed graph formed by the interconnection of a set of nodes ($N$) through the set of links ($L$) by the maps $\text{init}(L) : L \rightarrow N$ and $\text{ter}(L) : L \rightarrow N$, that assign to each edge an initial and terminal vertex respectively and will be written as $G_{\text{network}} = (N, L)$.

**Dynamical Networks**

Having defined networks generally we now focus specifically on dynamical networks. In order to properly define a dynamical network we need a few preliminary definitions which we present now.

**Definition 2.3** (Dynamical Node). A dynamical node ($N_{\text{dyn}}$) is a system whose operation, in isolation, is governed by a set of dynamical equations that describe implicitly how the state of the system changes with time. Dynamical nodes are given by the general form:

$$
\begin{align*}
x_k(t) &= f_k(x_k(t), u_k(t)) + g_k(x_k(t), y_k(t)) \\
y_k(t) &= h_k(x_k(t))
\end{align*}
$$

where $x$ is the time derivative of the node state vector $x \in \mathbb{R}^n$ that contains the quantities needed to describe the operating condition of the node. $u \in \mathbb{R}^m$ is the local control vector that gives the values of the local control inputs used to control the operation of the dynamical node. $y \in \mathbb{R}^r$ is the local output that gives the values of the measurable output of the dynamical node. $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n$, $g(x) : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $h(x) : \mathbb{R}^n \rightarrow \mathbb{R}^r$ are the internal, control and output functions respectively that determine how the local states and control inputs cause the internal and output states of the dynamical node to evolve.

**Definition 2.4** (Interconnection). An interconnection ($L_{kj}$) is a directed physical or information theoretic link between two dynamical nodes ($N_k$ and $N_j$), where $\text{init}(L_{kj}) : L_{kj} \rightarrow N_k$ and $\text{ter}(L_{kj}) : L_{kj} \rightarrow N_j$. We define the value of an interconnection between two dynamical nodes ($N_k$ and $N_j$) as $(l_{kj}(x_k, y_j) : \mathbb{R}^{n_k} \times \mathbb{R}^{p_j} \rightarrow \mathbb{R}^{n_j})$.

With this framework we are now able to define a dynamical network.

**Definition 2.5** (Dynamical Network). A dynamical network $G_{\text{net,dyn}} = (N_{\text{dyn}}, L)$ is a network where the set of nodes ($N$) and the set of interconnections ($L$) are as given previously (cf. Def 2.3 and Def 2.4 respectively). We write $G_{\text{net}} = (N, L)$ to represent the dynamical network $G_{\text{net,dyn}}$ where we have dropped the dynamical subscript. Mathematically a dynamical network consists of nodes ($N_k$) with a set of internal states ($x_k \in \mathbb{R}^{n_k}$), a set of local control inputs ($u_k \in \mathbb{R}^{m_k}$), and a set of measurable outputs given by ($y_k \in \mathbb{R}^{p_k}$). We also have an associated set of internal dynamics ($f_k(x_k) : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{n_k}$), an associated set of control dynamics ($g_k(x_k, u_k) : \mathbb{R}^{n_k} \times \mathbb{R}^{m_k} \rightarrow \mathbb{R}^{n_k}$) and output function given by ($h_k(x_k) : \mathbb{R}^{n_k} \rightarrow \mathbb{R}^{p_k}$). Furthermore each dynamic node is connected to other dynamic nodes through nonlinear interconnections ($l_{kj}(x_k, y_j) : \mathbb{R}^{n_k} \times \mathbb{R}^{p_j} \rightarrow \mathbb{R}^{n_j}$). This gives the dynamical model of the $k$-th dynamical system in a dynamical network as:

$$
\begin{align*}
x_k &= f_k(x_k) + g_k(x_k, u_k) + \sum_j l_{kj}(x_k, y_j) \\
y_k &= h_k(x_k)
\end{align*}
$$

**Hybrid Dynamical Networks**

Hybrid dynamical networks allow the continuous time operation of a single node or link to be broken into discrete operating modes. An operating mode of a hybrid system is a distinct...
type of behaviour which is governed by a set of dynamics that is different from other modes of the system. In this way we are able to partition the operation of the dynamical nodes into discrete modes. This provides an elegant way of abstracting the continuous time dynamics of the network into a form that allows faults and other changes in the network dynamics to be accurately modelled. We proceed using definitions similar to those in [Zhu and Hill, 2008].

**Definition 2.6 (Hybrid Dynamical Network).** A hybrid dynamical network is a dynamical network where the dynamical nodes \( N \) and interconnections \( L \) have a hybrid characteristic which can be represented in the form:

\[
\begin{align*}
\dot{x}_k &= f_{kx_k}(x_k) + g_{kx_k}(x_k, u_{kx_k}) + \sum_j h_{jx_k}(x_k, y_j) \\
y_k &= h_{kx_k}(x_k)
\end{align*}
\]

for \( k \in 1, 2, \ldots, K \) where \( x_k \) is the switching signal taking values in:

\[ M_k = \{1, 2, \ldots, m_k\} \]

\( M_k \) is the set of operating modes of the system. Additionally, \( x_k \in \mathbb{R}^{m_k} \) is the state vector of the system, \( u_{kx_k} \in \mathbb{R}^{m_{ux_k}} \) is the input control vector to the system and \( y_{ikz} \in \mathbb{R}^{p_k} \) is the output vector of the system. \( f_{kx_k}, g_{kx_k} \) and \( h_{kx_k} \) are continuous.

The switching signal \( (x_k: \mathbb{R}_+ \rightarrow M_k) \) for the \( k \)-th hybrid system can be characterised by the switching sequence:

\[ \Sigma_k = \{(i_{k_0}, t_{k_0}), (i_{k_1}, t_{k_1}), \ldots, (i_{k_z}, t_{k_z}) | i_{k_z} \in M_k, z \in \mathbb{Z}\} \]

where \( x_{k_0}, t_{k_0} \) are the initial system state and time respectively of the \( k \)-th system and \( \Sigma_k \) is the set of nonnegative integers. We note that \( t_1 = t_2 = \ldots = t_{k} = t \), that is all systems have the same time. When \( t_{k} \in [t_{k_1}, t_{k_1+1}) \), \( \sigma_k = i_{k_1} \), we say the \( i_{k_1} \)-th subsystem of system \( k \) is active and the trajectory of the switched system \( (x_{k}(t)) \) is defined as the trajectory of the system \( (x_{i_{k_1}}(t)) \) with switching signal \( i_{k_1} \in M_k \).

When there is only a single system in the network we recover the usual definition for a single hybrid dynamical system. We assume that the state of the system is continuous and thus does not exhibit abrupt changes at the instant of switching. The entire internal, control and network dynamics of the network at time \( t \) can be characterised by a \( K \)-tuple \((\sigma_1(t), \sigma_2(t), \ldots, \sigma_K(t)) \) of \( \{M_1 \times M_2 \times \cdots \times M_K\} \).

**Hybrid Dynamical Modes**

We present here some definitions and concepts relevant to the representation of hybrid dynamical nodes as graphs in order to more clearly define what we mean by the modes of a hybrid dynamical network.

**Definition 2.7 (Set of States).** The set of states \( \{X\} \) of a dynamical node is a discrete set composed of the functions \( (x_1 : \mathbb{R}_+ \rightarrow \mathbb{R}) \) that for each \( t \geq 0 \) returns the value of the node state defined by the function \( x_1 \).

**Definition 2.8 (Set of Outputs).** The set of outputs \( \{Y\} \) of a dynamical node is a discrete set composed of the functions \( (y_1 : \mathbb{R}_+ \rightarrow \mathbb{R}) \) that for each \( t \geq 0 \) returns the value of the node output defined by the function \( y_1 \).

**Definition 2.9 (Set of Controls).** The set of controls \( \{U\} \) of a dynamical node is a discrete set composed of the functions \( (u_1 : \mathbb{R}_+ \rightarrow \mathbb{R}) \) that for each \( t \geq 0 \) returns the value of the node controls defined by the function \( u_1 \).

We now define our dynamical node as a graph progressing in a similar but slightly more general way than was presented in [Boukhobza et al., 2007].

**Definition 2.10 (Dynamical Node Graph).** A dynamical node graph is a directed graph representing the internal interconnection links between states for the \( k \)-th dynamical node is given by \( L_{XX_k} = \{(u_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1, k_2 \in 1, \ldots, m_k \). The internal interconnection links between control inputs and states for the \( k \)-th dynamical node is given by \( L_{UX_k} = \{(u_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \). The internal interconnection links between states and outputs for the \( k \)-th dynamical node is given by:

\[ L_{XY_k} = \{(x_{k_1_1}, y_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \]

\( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \) and the interconnection links between all the network outputs and internal states for the \( k \)-th dynamical node is given by \( L_{YX_k} = \{(y_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \) where \((v_1, v_2)\) denotes a directed edge from node \( v_1 \in N \) to node \( v_2 \in N \).

We also define the capacity of a structural dynamic graph as a measure of the magnitude of each of the non-zero links defined in the link sets above.

**Definition 2.11 (Set of Links).** The internal interconnection links between states for the \( k \)-th dynamical node is given by \( L_{XX_k} = \{(x_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1, k_2 \in 1, \ldots, m_k \). The internal interconnection links between control inputs and states for the \( k \)-th dynamical node is given by \( L_{UX_k} = \{(u_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \). The internal interconnection links between states and outputs for the \( k \)-th dynamical node is given by:

\[ L_{XY_k} = \{(x_{k_1_1}, y_{k_1_2}) | \sigma_{k_1_1} \neq 0\} \]

\( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \) and the interconnection links between all the network outputs and internal states for the \( k \)-th dynamical node is given by \( L_{YX_k} = \{(y_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( k_1 \in 1, \ldots, m_k \), \( k_2 \in 1, \ldots, m_k \) where \((v_1, v_2)\) denotes a directed edge from node \( v_1 \in N \) to node \( v_2 \in N \).

We also define the capacity of a structural dynamic graph as a measure of the magnitude of each of the non-zero links defined in the link sets above.

**Definition 2.12 (Capacity of a Dynamical Node Graph).** The capacity of a structural dynamic graph is a set \( C_{L_k} \) that assigns to each link \( L_i \in L \) a measure of the capacity of that link. For the link sets defined in Def. 2.11 we have the following capacity sets for the \( k \)-th dynamical node: \( C_{L_{XX_k}} = \{(x_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( L_{UX_k} = \{(u_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \), \( L_{XY_k} = \{(x_{k_1_1}, y_{k_1_2}) | \sigma_{k_1_1} \neq 0\} \) and \( C_{L_{YX_k}} = \{(y_{k_1_1}, x_{k_1_1}) | \sigma_{k_1_1} \neq 0\} \).

From these definitions we are able to very precisely define the concept of a change in mode and this is the topic of the next section.

**Mode Changes**

A change in mode is a change of the fundamental governing dynamics and we can see this as follows. For the switching sequence of the \( k \)-th node in a hybrid dynamical network, \( \Sigma_k = \{(i_{k_0_1}, t_{k_0_1}), (i_{k_1_1}, t_{k_1_1}), \ldots, (i_{k_z_1}, t_{k_z_1}) | i_{k_z_1} \in M_k, z \in \mathbb{Z}\} \) we take \( L_{ik_z} = L_{X_{ik_z}} \cup L_{UX_{ik_z}} \cup L_{Y_{ik_z}} \cup L_{XY_{ik_z}} \) to be the total link set in the time interval \([t_{k_z_1}, t_{k_z_1+1})\) and \( C_{L_{ik_z}} = C_{L_{XX_{ik_z}}} \cup C_{L_{UX_{ik_z}}} \cup C_{L_{XY_{ik_z}}} \cup C_{L_{YX_{ik_z}}} \) to be
the equivalent total capacity set in the same interval. A mode change implies that for any \( z \in \mathbb{Z} \), \( L_{ikz} \neq L_{ikz+1} \) and/or \( C_{L_{ikz}} \neq C_{L_{ikz+1}} \). That is the change in mode forces the link and/or capacity sets to change, meaning a fundamental change in the fundamental governing dynamics. It should be noted that determining the link and capacity sets is possible using a variety of parameter and system identification methods, however, these methods are beyond the scope of this paper. With these concepts fully defined we are now able to understand how we can unify hybrid dynamical networks and discrete diagnosis.

2.2 Model Based Diagnosis

Diagnosis is the problem of determining what possibly happened on a system given some observations of its behaviour. In model-based diagnosis, we assume that we have a model that describes the discrete transitions between modes in the system (i.e. given an operational mode the model describes the possible transitions that will result in a new operational mode). The model is an abstraction of the system transitions and is often represented by an automaton.

2.3 Diagnosis of Discrete Event Systems

We outline the diagnosis of discrete event systems in order to apply this approach to the diagnosis of hybrid dynamical networks. Essentially, we perform an abstraction on the discrete part of a hybrid system to obtain a model on which we can perform diagnosis. More details are given in section 3.2.

We consider the discrete evolution of a system. The set of all possible behaviours of the system is a language denoted \( \text{Mod} \) over the set of events \( \Sigma \) that could possibly occur on the system. In our work, we assume that observations we make are on the modes of a given system (in contrast to more commonly used event-based observations [Largouet and Cordier, 2000]). We denote these observations (a set of possible modes) on the system by a language \( \text{Obs} \). The diagnosis of the system can be computed as

\[
\Delta = \text{Mod} \otimes \text{Obs},
\]

where \( \otimes \) is the projection of the model onto the observations. The diagnosis \( \Delta \) essentially determines the current operational mode consistent with the observations. In this way, we are able to refine the original mode estimates in \( \text{Mod} \) and/or \( \text{Obs} \).

2.4 Distributed Diagnosis

It is possible to use diagnosis techniques to overcome the problem of diagnosing large and complex systems which is of considerable interest. We make use of distributed diagnosis algorithms on which one of the authors is working [Kan John and Grastien, 2008] to minimise the computational complexity in large diagnosis problems.

In a distributed diagnosis approach [Pencolé and Cordier, 2005], [Cordier and Grastien, 2007], [Su and Wonham, 2005], [Kan John and Grastien, 2008] we only consider local computations. Rather than considering the diagnosis on the whole network, we only compute diagnoses locally (on each node). The problem is then to make sure that the local sets of diagnoses are globally consistent. Global Consistency

We consider a network \( S \) of interconnected nodes \( S = \{S_1, S_2, \ldots, S_k\} \). The diagnosis of \( S_i \) is given by

\[
\Delta_i = \text{Mod}_i \otimes \text{Obs}_i,
\]

The diagnosis \( \Delta \) of \( S \) is said to be globally consistent if the overall diagnosis agrees with the diagnosis at each node. A strict definition of global consistency is presented in [Kan John and Grastien, 2008]. Local (pairwise) consistency does not normally ensure global consistency. However, we can transform the topological graph of the whole network into a form (in our case a junction tree) which will ensure that local consistency leads to global consistency [Kan John and Grastien, 2008]. We then have groups of nodes as clusters of the tree. Diagnosis is performed locally on each cluster and local consistency is applied until a fixpoint is reached.

3 Unifying Discrete Diagnosis and Hybrid Dynamical Networks

We explain in this section the method by which hybrid dynamical networks and discrete diagnosis may be unified. We first discuss some of the assumptions that we are making during this analysis.

3.1 Analysis Assumptions and Discussion

The assumptions used in this analysis are ideally kept to a minimum to maximise the real-world applications of this approach. Importantly these assumptions do not restrict the method by which a mode changes. It is reasonable to assume that an event may be due to a fault, a change in environmental conditions, the effects of a control choice in a given mode or a variety of other reasons. By looking at the outcome of the event rather than the source of the event we are better placed to use this methodology for dealing with many of the challenges faced in modern dynamical networks. Our assumptions also allow faults to occur in succession provided they do not occur instantaneously. This is also advantageous as it allows us to deal with cascading type faults which are of significant concern in modern hybrid dynamical networks.

The following assumptions are made during the analysis.

Mutual Exclusivity of Modes

We assume that the modes \( (M_k) \) of a system are mutually exclusive. That is a system \( (S_k) \) in the dynamic network can only be in one mode at any time. That is if \( \sigma_k = m_i \in M_k \) for \( t_k \in [t_k_i, t_k_i+1) \) and \( \sigma_k = m_j \in M_k \) for \( t_k \in [t_k_i, t_k_i+1) \) then \( m_i \neq m_j \). This is a natural assumption that ensures any events cause the system to move into one of a discrete set of modes and that one and only one mode is responsible for describing the dynamic evolution of a system in the network at a given time.

Timing of Events

We assume that between two observations only one event can occur. This further implies that for any finite \( T > t_0 \), there...
exists a positive integer $K_T$, which may depend on $T$, such that during the time interval $[t_0, T]$ each dynamical node (Eq. 1) switches no more than $K_T$ times. This ensures that it is practically possible to diagnose a system as the sequence of observations will have information about all the faults that occur in the system.

**Controller Laws**
We are assuming that for a given operational mode it is possible to compute an appropriate controller off-line [Khalil, 2002]. The methods for achieving this are too many and varied to enumerate and cover the span of classical and modern control for a period of almost a century. For this reason we focus purely on the method of choosing an appropriate control strategy when the fault occurs. A simple justification of this can be seen in that if the operating mode of each dynamical node can be chosen independently then there are:

$$
\prod_k m_k
$$

K-tuples that characterise the operating condition of the hybrid dynamical network. In networks with large numbers of systems and internal modes this rapidly becomes computationally intractable.

In the majority of dynamical networks a single event or fault will effect a number of systems in the network and thus using diagnosis methodologies allows this number to be drastically reduced so that the off-line computation of controllers is actually feasible. Furthermore if we consider networks where only a subset of the systems are directly observed then without the relationships stored in the diagnosis algorithm it would be impossible to diagnose and control for the effects of those unobserved systems.

**Mode Determination**
Again, the ability to determine the modes is beyond the scope of this paper and covers methodologies in system and parameter identification [L Ung, 1999] [Blackhall and Rotkowitz, 2008] over the last half century or so. The authors believe strongly that it is a more than valid assumption that modes can be determined with high accuracy in the majority of scenarios.

### 3.2 Diagnosis of Hybrid Dynamical Networks

In diagnosis terms we consider a hybrid dynamical network $S$ of interconnected dynamical nodes $S = \{S_1, S_2, \ldots, S_K\}$. This means that each node has a defined operating mode and each transition causes a change in mode that redefines the continuous behaviour of the system.

We can think of each system in $S$ as having a continuous behaviour captured by a tuple $S_{k,\text{cont}} = (x_k, \dot{x}_k, y_k, \dot{y}_k, u_{k,1}, \ldots, u_{k,K})$ where the elements of the tuple have been defined previously.

We also make an abstraction on the discrete behaviour of each dynamical node using an automaton. Thus the automaton representing the discrete behaviour of a system can be given by the tuple $S_{k,\text{disc}} = (\mathcal{M}_k, \Omega_k, T_k, \sigma_{k,0})$.

- $\mathcal{M}_k$ is the finite set of system modes $(m_1, m_2, \ldots, m_K)$.
- $\Omega_k$ is the set of events $(\omega_1, \omega_2, \ldots, \omega_l)$ which determine switching between modes.
- $T_k$ is the transition function that maps an event and mode into a new mode, $T : \Omega_k \times \mathcal{M}_k \rightarrow \mathcal{M}_k$.
- $\sigma_{k,0}$ is the initial mode of each system.

Using the diagnosis methods defined previously in sections 2.3 and 2.4 it is possible to compute a diagnosis of the discrete behaviour of each dynamical node. As each discrete behaviour has a corresponding continuous behaviour we have thus shown that it is possible to perform diagnosis of a continuous hybrid dynamical network using the discrete model based diagnosis methodology.

### 3.3 Algorithmic Considerations

We outline two methods in which distributed diagnosis can be used in maintaining a fault tolerant network. Algorithm 1 presents a global optimal approach. Algorithm 2 utilises the power of the junction tree technique to address the time criticality issue that often arises in large hybrid dynamical networks to try and avoid cascading failures.

We consider a hybrid dynamical network $S = \{S_1, S_2, \ldots, S_K\}$ each of which has an automaton model. We thus have a set of models $Mod_1, \ldots, Mod_K$. We can then use a set $C = \{C_1, \ldots, C_l\}$ of controllers that can be used on the system. At a given time $t$, we have observations $Obs_1, \ldots, Obs_K$ on the system where each $Obs_k$ is a set of estimates of the mode of a given component $S_k$ at $t$ obtained from parameter estimation and system identification.

#### Algorithm 1: Global Optimal Algorithm for diagnosis and control of a hybrid dynamical network

1. At time $t_{obs}$
2. input $\{Mod_1, \ldots, Mod_K\}, \{Obs_1, \ldots, Obs_K\}, C$
3. for all $t_{obs} \in [t_0, \ldots, t_{final} do$
4. do $\Delta_{t_{obs}} := Mod_{t_{obs}} \odot Obs_{t_{obs}}$
5. end for
6. run consistency algorithm (returns globally consistent network)
7. for all $t_{obs} \in [t_0, \ldots, t_{final} do$
8. pick control strategy $C$ from $C$
9. end for

Algorithm 1, we pick the best control strategy over a globally consistent network. Provided that the diagnosis and global consistency can be performed fast enough, this provides the global optimal solution. However, in a large network, time is often a critical factor in the occurrence of cascading type faults. If we can choose a locally optimal controller after performing local consistency, this could help prevent other dynamical nodes in the system from failing as a potential fault is captured and dealt with as soon as possible. This approach is outlined in Algorithm 2.

### 3.4 Diagnosis under partial observations

The problem with large networks is that it is not always possible or practical to obtain observations on all dynamical nodes. However, because components share common events, it is
Algorithm 2 Optimal Time Critical Algorithm for diagnosis and control of a hybrid dynamical network

1: At time $t$
2: input $\{Mod_1, \ldots, Mod_K\}, \{Obs_1, \ldots, Obs_K\}, C$
3: for all $t_{obs} \in t_0, \ldots, t_{final}$ do
4: do $\Delta_i := Mod_i \otimes Obs_i$
5: end for
6: while not globally consistent do
7: Perform local consistency with neighbours
8: pick a control strategy $C$ from $C$
9: end while

still possible to diagnose unobserved nodes by using the diagnosis of the observed ones. For an unobserved component, we take $Obs$ to be the set of all possible modes. We then refine the diagnosis when performing local consistency with other components.

4 Numerical Example

We will use the fault diagnosis methodology developed to diagnose and recover from failures on a four node complex network (Fig. 1). Complex networks (described for example in [Li et al., 2004]) have emerged out of the biological and social sciences and are an attempt to explain the behaviour of large interconnected groups. From a dynamical and systems theoretic perspective complex networks consist of independent dynamical nodes connected together. We use the following representation:

$$\dot{x}_i = f(x_i) + c \sum_j a_{ij} \Gamma (x_j - x_i) + \epsilon_i (s(t) - x_i)$$

where it is clear that this complex network is in the dynamical network form presented in Eq. 2. We have $x \in \mathbb{R}^n$ is the state vector, $f(x): \mathbb{R}^n \to \mathbb{R}^n$ is the unique internal dynamics, $a_{ij}$ determines the interconnections in the network and is symmetric. As such $a_{ij} = a_{ji} = 1$ if node $i$ and node $j$ are connected together and zero otherwise. $\Gamma$ is the feedback connectivity matrix that determines how the difference vector $(x_j - x_i)$ affects the internal state and $c$ is the scalar gain that determines the magnitude of this effect. We also have the control input $\epsilon_i(s(t) - x_i)$ where $s(t)$ is the desired equilibrium state of the dynamical node and the network as a whole and $\epsilon_i \in \{0, \epsilon\}$. That is only some nodes have a local feedback control applied with control gain $\epsilon$. This is referred to in the literature as pinning control [Li et al., 2004]. In complex networks the control problem becomes ensuring that the network will synchronise, where synchrony is defined as:

$$x_1 = x_2 = \cdots = x_n \to s(t)$$

such that $\dot{s}(t)) = f(s(t))$ is the solution of the isolated nodes.

It is well known in the complex networks community that the ability of the network to synchronise is dependent on the value of $c$ chosen as well as the location and magnitude of the pinning control that is applied to the network. Whilst an arbitrarily large $c$ and pinning control gain ($\epsilon$) can always be chosen this is often physically impossible as well as being inefficient as much more control authority is being commanded than is necessary to achieve the desired synchrony.

In our example we choose $f(x_i) = -x_i^2 + 5x_i$, $\Gamma = I$, the four dimensional identity matrix and the network is globally connected. With $c = 1.5$ we can see the resultant state trajectory in Fig. 2, where synchrony is rapidly achieved.

We consider faults that effect the structure of the network (similar to [Kim and Hill, 2008]). That is we consider a network where the value of $a_{ij} = a_{ji}$ changes when a fault occurs, thus representing the hybrid component of the dynamical network presented in Eq. 7. This further implies that only the link sets change when a fault occurs ($L_{ik} \neq L_{ik+1}$). That is the fault causes a fundamental change in the network dynamical structure. The capacity sets do not change($C_{L_{ik}} \cap C_{L_{ik+1}} \subseteq C_{L_{ik+1}}$) and thus networks of this type are said to be capacity invariant. Even in this simple network the possible structural dynamical fault modes are substantial. We consider ten possible events, four due to node removal and six due to link removal. In the case of node removal we assume that all the links between the removed node and the network are also removed simultaneously. These fault modes correspond to the typical faults found in complex network operation. We consider two possible fault scenarios in this example:

- The network link $L_5$ becomes disconnected.
- The network system $N_4$ becomes disconnected.

We assume that we only have partial observations on the network and thus we are only able to monitor the modes of systems $N_1$ and $N_2$. The two faults outlined are important because it is only when we use the diagnosis algorithm, and the observations from both nodes $N_1$ and $N_2$, that we are able to exactly determine the fault that occurred and implement an appropriate control strategy.

While we can always assume that we can increase the control effect on the nodes in the case of a fault, this is an inappropriate action as it is known that when network link $L_5$ becomes disconnected no additional control action is required to ensure network synchrony. Conversely, when network sys-
Figure 2: The state trajectories of each system in the network when the network is globally connected.

Figure 3: The state trajectories of each system in the network when the network link $L_5$ becomes disconnected due to a fault. The diagnosis algorithm is used here to determine the fault that has occurred and that no additional control input is required to achieve network synchrony.

Figure 4: The state trajectories of each system in the network when the system $N_4$ becomes disconnected from the network due to a fault. The diagnosis algorithm is not running and we do not initiate any external control.

diagram.
Figure 5: The state trajectories when the node $N_4$ becomes disconnected from the network due to a fault. The diagnosis algorithm detects the fault and switches in a local control input on node $N_1$ to ensure network synchrony.

the network performance objectives to be achieved. The optimal location of observation and control is still an area of active research.

5 Conclusion and Future Work

In this paper we have shown how it is possible to use model based diagnosis to create a fault tolerant dynamical network. We have outlined the method by which the two concepts can be unified and motivated the development of this algorithm by showing how this algorithm will offer a substantial improvements in the fault diagnosis and recovery of dynamical networks. Furthermore we have shown how this particular approach will result in the ability to do globally consistent distributed diagnosis. We believe that this methodology is an important advance in the unification of the study of hybrid dynamical networks and advanced, computationally feasible analysis tools. This paper is intended to lay the ground work for the development of advanced algorithms capable of being used in real world scenarios.

The most important area of future work is the implementation of this algorithm with real systems and data. It is likely that implementations will focus on electricity networks as this represents the most urgent need for algorithms and methodologies of this type.

6 Acknowledgements

The authors gratefully acknowledge the guidance, support and advice of Dr Alban Grastien and Professor David Hill.

References


An Improved Approach for Generating Max-Fault Min-Cardinality Diagnoses

Johan de Kleer
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA

Abstract

Most approaches to model-based diagnosis focus on isolating defective component(s) by performing additional measurements on the defective system. Sometimes internal measurements are expensive to make and it is much less costly to change system inputs and observe how outputs change. In digital circuits this is called test-vector generation. Of particular interest are Max-Fault Min-Cardinality (MFMC) observation vectors which result in the maximum number of faults in the minimal cardinality diagnosis. Prior approaches to MFMC generation either used sampling (which is incomplete) or exhaustively enumerate all possible observation vectors (which is computationally impossible). This paper presents a new direct approach to determining MFMC vectors which shows 4-5 orders of magnitude performance improvement over prior algorithms.

1 Introduction

This paper provides a new approach to identifying Max-Fault Min-Cardinality (MFMC) observation vectors as defined in [Feldman, Provan, & van Gemund, 2007]. Identifying such vectors has widespread applicability to analyzing the diagnosability of systems and evaluating the scalability of diagnostic algorithms. Most existing algorithms work best with single or double faults, but do not scale well to higher cardinalities. MFMC observation vectors can easily yield minimal diagnoses of size 20 and above. The main result of this paper is that MFMC observation vectors can be generated with familiar algorithms and well-known circuit properties at 4-5 orders of magnitude performance over prior algorithms.

We first explain the concepts intuitively in an example. Consider the circuit illustrated in Figure 1. The digital circuit consists entirely of NAND gates. Figure 2 presents the truth table for a NAND gate.

Suppose the inputs are observed to be I1=1, I2=0, I3=0, I4=0 and I5=0. Given those inputs the outputs must be O1=1 and O2=0. Consider the subset of the circuit consisting of G3, G4 and G6. Given I3=0, G3 drives N3=1. Given I5=0, drives N4=1. Given both inputs to the NAND gate G6 are 1, it drives its output to O2=0. Suppose O2=1 is observed. This can only

Figure 1: The simplest circuit, c17, from the ISCAS-85 test suite. Inputs are labeled “In,” outputs “On,” gates “Gn,” and corresponding internal nodes “Nn.” All 6 gates are NAND gates.

Figure 2: Truth table for NAND gate.
be explained by at least one of G3, G4 and G6 being faulted. Consider the subset of the circuit consisting of G1, G2 and G5. Given I2=0, G1 drives N1 to 1. Given both N1 and I1 are 1, G2 drives N2 to 0. If one input to a NAND is 0, the output is driven to 1 (O1). Suppose we observe O1=0. Then at least one of G1, G2 or G5 is faulted. The combined observations can only be explained by a double fault (consisting of one of G1,G2 and G5 combined with one of G3, G4 and G6). These are all minimal diagnoses. The observation is an MFMC vector as no other observation has a larger min-cardinality diagnosis. (This will be shown later.)

Prior approaches to identifying MFMC observations utilized either importance sampling, simulated annealing or exhaustive search of the entire space. The first two approaches are not exhaustive and cannot produce absolute bounds. Exhaustive search is impractical for all but the smallest circuits because the complexity is $2^{\text{INPUTS}+\text{OUTPUTS}+\text{COMP}}$. This paper proposes an alternative approach to analyze the circuit structure to greatly lower this complexity. This approach has been completely implemented and is based on the GDE implementation and prime implicate algorithms described in [Forbus & de Kleer, 1992]. The result significantly outperforms the previous approach. In future work, we propose to use a more efficient approach outlined in [de Kleer & Williams, 1989] from which we expect further dramatically improved performance.

In this paper we draw all our examples from digital circuits. However, the concepts apply to any system which can be modeled using discrete signals and having distinguished inputs and outputs.

2 Formal Framework

We adopt the framework of [de Kleer & Williams, 1987] which we briefly summarize.

**Definition 1** A system is a triple $(SD, COMP, OBS)$ where:
1. $SD$, the system description, is a set of first-order sentences.
2. $COMP$, the system components, is a finite set of constants.
3. $OBS$, a set of observations, is a set of first-order sentences.

**Definition 2** Given two sets of components $Cp$ and $Cn$ define $D(Cp, Cn)$ to be the conjunction:
$$\left(\bigwedge_{c \in Cp} AB(c)\right) \land \left(\bigwedge_{c \in Cn} \neg AB(c)\right).$$

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 3** Let $\Delta \subset COMP$. A diagnosis for $(SD, COMP, OBS)$ is $D(\Delta, COMP - \Delta)$ such that the following is satisfiable:
$$SD \cup OBS \cup \{D(\Delta, COMP - \Delta)\}$$

**Definition 4** An $AB$-literal is $AB(c)$ or $\neg AB(c)$ for some $c \in COMP$.

**Definition 5** An $AB$-clause is a disjunction of $AB$-literals containing no complementary pair of $AB$-literals.

**Definition 6** A conflict of $(SD, COMP, OBS)$ is an $AB$-clause entailed by $SD \cup OBS$.

**Definition 7** A minimal conflict of $(SD, COMP, OBS)$ is a conflict no proper sub-clause of which is a conflict of $(SD, COMP, OBS)$.

**Theorem 1** Suppose that $\Pi$ is the set of minimal conflicts of $(SD, COMP, OBS)$, and that $\Delta$ is a minimal set such that,
$$\Pi \cup \left\{ \bigwedge_{c \in COMP - \Delta} \neg AB(c) \right\}$$

is satisfiable. Then $D(\Delta, COMP - \Delta)$ is a minimal diagnosis.

The lemma forms the basis of most model-based diagnosis algorithms: (1) compute the minimal conflicts, (2) compute the diagnoses. For the purposes of MFMC computation we are only interested in minimal diagnoses:

**Definition 8** A diagnosis $D(\Delta, COMP - \Delta)$ is a minimal diagnosis iff for no proper subset $\Delta' \subset \Delta$ is $D(\Delta', COMP - \Delta')$ a diagnosis.

We also assume the usual axioms for equality and arithmetic are included in SD. For this paper, we assume weak fault models or the Ignorance of Abnormal Behavior property.

3 Representing Component Behaviors as Propositional Clauses

An inverter can be modeled by:
$$INVERTER(x) \rightarrow \left[\neg AB(x) \rightarrow [in(x) = 0 \equiv out(x) = 1]\right].$$

A particular inverter $G$ is thus modeled by the formula
$$\neg AB(G) \rightarrow [in(G) = 0 \equiv out(G) = 1],$$

which is modeled by the following clauses (prime implicates):
$$AB(G) \lor out = 0 \lor in = 1,$$
$$AB(G) \lor out = 1 \lor in = 0.$$

The NAND gate $G5$ (of Figure 1) with inputs N2 and N3 and output O1 is modeled by the clauses:
$$AB(G5) \lor N2 = 0 \lor N3 = 0 \lor O1 = 0,$$
$$AB(G5) \lor N2 = 1 \lor O1 = 1,$$
$$AB(G5) \lor N3 = 1 \lor O1 = 1.$$
4 MFMC definitions

Following [Feldman, Provan, & van Gemund, 2007] we define:

Definition 9 The cardinality of diagnosis \( \mathcal{D}(\Delta, \text{COMP} \text{S} - \Delta) \) is \(|\Delta|\).

Definition 10 A diagnosis \( \mathcal{D}(\Delta, \text{COMP} \text{S} - \Delta) \) is a minimal cardinality diagnosis if for no other diagnosis \( \mathcal{D}(\Delta', \text{COMP} \text{S} - \Delta') \) is \(|\Delta'| < |\Delta|\).

Clearly all minimal cardinality diagnoses are minimal diagnoses. However, the converse does not hold.

Definition 11 MaxCard(\(SD\)) is the maximum cardinality of all possible minimal cardinality diagnoses for all possible OBS.

Definition 12 An MFMC observation is any OBS such that the minimal cardinality diagnosis is equal to MaxCard(\(SD\)).

In this paper we will assume the usual definitions of well-formedness (system forms a DAG, only one component drives any node, all component inputs are driven, components have at least one input and at most one output).

We can now state the analysis of the circuit of Figure 1 more formally. \(\text{COMP} \text{S} = \{G_1, G_2, G_3, G_4, G_5\}\). \(\text{OBS} = \{I_1 = 1, I_2 = 0, I_3 = 0, I_4 = 0, I_5 = 0, O_1 = 0, O_2 = 1\}\). There are 3 minimal conflicts:

\[
\begin{align*}
&AB(G_4) \lor AB(G_5) \lor AB(G_6), \\
&AB(G_1) \lor AB(G_2) \lor AB(G_5), \\
&AB(G_3) \lor AB(G_4) \lor AB(G_6).
\end{align*}
\]

One of the minimal diagnoses is:

\[
\neg AB(G_1) \land \neg AB(G_2) \land AB(G_3) \land \\
\neg AB(G_4) \land AB(G_5) \land \neg AB(G_6).
\]

5 Two Bounds on MaxCard

There are two important upper bounds on MaxCard. The first is obvious, but important:

Theorem 2 Max\text{Card}(SD) is bounded by \(|\text{COMP} \text{S}|\).

More importantly:

Theorem 3 Max\text{Card}(SD) is bounded by the number of outputs of SD.

Proof sketch. In a well-formed circuit, every possible input combination is possible. Thus any conflict must involve at least one component driving an output. So the conjunction of all output components faulted will always be a diagnosis. It may not be minimal and thus the number of outputs is only an upper bound on Max\text{Card}(SD).

The “converse” is not true: Max\text{Card}(SD) is not bounded by the number of inputs. Unfortunately, Max\text{Card}(SD) may be far less than the number of outputs. Nevertheless, this second bound is critical to the efficiency of the algorithm described later: when searching for possible MFMC observations, those with a number of incorrect outputs less than the current best estimate of Max\text{Card}(SD) can be skipped.

\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
obs & MC & obs & MC \\
\hline
11111 & 0 & 01111 & 1 \\
11110 & 1 & 01110 & 0 \\
11101 & 1 & 01101 & 2 \\
11100 & 2 & 01100 & 1 \\
11011 & 2 & 01101 & 0 \\
11010 & 1 & 01010 & 1 \\
11001 & 1 & 01011 & 1 \\
11000 & 0 & 01000 & 2 \\
10111 & 2 & 00111 & 0 \\
10110 & 1 & 00110 & 1 \\
10101 & 1 & 00101 & 1 \\
10100 & 0 & 00100 & 2 \\
10011 & 1 & 00011 & 2 \\
10010 & 2 & 00010 & 1 \\
10001 & 0 & 00001 & 1 \\
10000 & 1 & 00000 & 0 \\
\hline
\end{tabular}
\caption{Minimum cardinality for each possible observation vector for subtractor circuit. Each obs = \{x, y, p, b, d\}.}
\end{table}

6 Sources of Complexity

A brute force approach to determining Max\text{Card}(SD) is to use a diagnosis engine to compute a minimal cardinality diagnosis for each possible observation. Consider the subtractor circuit of Figure 3 [Feldman, Provan, & van Gemund, 2007] having 7 components, 3 inputs and 2 outputs. Table 1 lists the minimum cardinality for the given observation vector. The Max\text{Card}(SD) for the circuit is 2 which is the upper bound stipulated by Theorem 3.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig3}
\caption{Subtractor with inputs x, y and p and outputs d and b. \(h_1\) and \(h_2\) are exclusive-or gates, \(h_3\) and \(h_5\) are OR gates, \(h_4\) and \(h_7\) are AND gates, and \(h_6\) is an inverter.}
\end{figure}

Although C17 has fewer components it is more complicated for a brute force algorithm. It has 5 inputs, 2 outputs, and 6 gates. The space to be searched for identifying Max\text{Card}(c17) is \(2^5 + 2^2 + 6 = 8192\) cases.

7 Reducing the number of components

The complexity of searching for Max\text{Card}(SD) is exponential in the number of components. Therefore, any reduction in number of components which does not affect
MaxCard(SD) provides significant computational advantage. Consider again the circuit of Figure 1. Gates G2 and G5 can be combined into one gate G2-G5 with one output O1 and three inputs I1, N1 and N3. This combination corresponds to the top dashed region of Figure 4. The only other combinable components is the bottom region. No other combination is possible.

The intuition why G2 and G5 can be combined is as follows. The internal node N2 is not observable. Therefore, any conflict involving G2 will necessarily involve G5 as well. If I1=1 and N1=1, then if G2 and G5 were operating correctly O1=1. Observation O1=0 yields the conflict \( \{G2, G5\} \). (Notice that G2 does not necessarily occur in every conflict involving G5. If N3=0, observing O1=0, yields the conflict \( \{G5\} \).) The same pattern arises in every possible conflict involving G2. Therefore, for the purposes of calculating MaxCard(SD), one can replace G2 and G5 with one composite component thereby reducing the complexity of searching for MaxCard(SD).

In the case of stuck-at faults, there are many algorithms to collapse faults [Bushnell & Agrawal, 2000]. In general:

**Theorem 4** Any connected subset of components of SD having only one output can be replaced by a single composite component with equivalent behavior without changing MaxCard(SD).

Identifying all such subsets is typically too expensive. Fortunately, many such equivalent subsets can be identified very quickly:

**Theorem 5** If component A drives internal node n and that node is an input into only one component B, then A, B and n can be replaced with a single composite component with logically equivalent behavior without changing MaxCard(SD).

Proof sketch. By construction, the input-output behavior of the circuit remains unchanged with the replacement. As no other component is connected to n every conflict involving A must also include B. Therefore, B would appear in any diagnosis in which A appears. Thus replacing A and B with a single composite component does not change MaxCard(SD).

For example component G2, node N2 and component G5 of Figure 4 can be replaced with a single component with inputs I1, N1 and N3 and output O1.

By iteratively applying the preceding theorem the number of components can be significantly reduced without changing MaxCard(SD).

The combined G2-G5 (G) component is described by the following 4 clauses (a single NAND gates is encoded by 3 clauses):

\[
AB(G) \lor N1 = 1 \lor N3 = 0 \lor O1 = 0,
\]

\[
AB(G) \lor I1 = 1 \lor N3 = 0 \lor O1 = 0,
\]

\[
AB(G) \lor N1 = 0 \lor I1 = 0 \lor O1 = 1,
\]

\[
AB(G) \lor N3 = 1 \lor O1 = 1.
\]

Table 2 lists the reduction in component count after applying the reduction rule. The circuits are commonly known 74nnn circuits or are from the ISCAS 85 [Brglez & Fujisawa, 1985] benchmarks. We use the prime implicate algorithm from [Forbus & de Kleer, 1992] on the clausal form of the component models.

The cones formulation of [Siddiqi & Huang, 2007] yields an analogous reduction in components. We cannot compare the timings of the underlying diagnostic algorithms or the resulting algorithms to compute MFMC vectors directly. Our current algorithm cannot complete the MFMC computation for any of the non-trivial ISCAS benchmarks. However, we do know lower bounds on all the benchmarks from partial runs and they typically scale with the number of outputs. For example, C2670 has 140 outputs and an MFMC of at least 20 (probably much higher).

### 8 Reducing the Number of Inputs and Outputs

Important properties which improve running time dramatically on many circuits are:

- Suppose SD can be divided into two SD1 and SD2 such they share no internal nodes. In this case, MaxCard calculation is greatly simplified: MaxCard(SD) = MaxCard(SD1) + MaxCard(SD2).

---

Table 3: The columns are: circuit, original complexity, MaxCard(SD), our wall time, cited wall time reported. Timings are on Common Lisp and a multi-core 3GHz PC and include prime implication construction time.

<table>
<thead>
<tr>
<th>circuit</th>
<th>complexity</th>
<th>MC</th>
<th>time</th>
<th>cited</th>
</tr>
</thead>
<tbody>
<tr>
<td>7485</td>
<td>$1.4 \times 10^{14}$</td>
<td>3</td>
<td>1ms</td>
<td>196.9s</td>
</tr>
<tr>
<td>74181</td>
<td>$4.0 \times 10^{16}$</td>
<td>7</td>
<td>60m</td>
<td>impossible</td>
</tr>
<tr>
<td>74182</td>
<td>$8.6 \times 10^{9}$</td>
<td>5</td>
<td>1ms</td>
<td>53.4s</td>
</tr>
<tr>
<td>74283</td>
<td>$1.1 \times 10^{14}$</td>
<td>5</td>
<td>40ms</td>
<td>371.9s</td>
</tr>
<tr>
<td>c17</td>
<td>$8.2 \times 10^{6}$</td>
<td>2</td>
<td>2ms</td>
<td>not given</td>
</tr>
</tbody>
</table>

- Let $SD|_{i=x}$ represent the system $SD$ with input $i$ set to $x$ and the resulting circuit simplified by removing irrelevant components. $MaxCard(SD) = Max(MaxCard(SD|_{i=0}), MaxCard(SD|_{i=1}))$.
- If all of the inputs of a components are fed directly from the inputs (and no other component is driven by those inputs), then this set of inputs can be reduced to one and the component replaced by a buffer.
- Under many conditions, inputs can be discarded without affecting $MaxCard(SD)$. For example, $F_1$ can be discarded from $c17$ and $G_2$ replaced by an inverter.

9 Basic Algorithm

The algorithm is based on the Common Lisp code of [Forbus & de Kleer, 1992] and exploits all the prior concepts described earlier.

1. Inputs and outputs are reduced.
2. The number of components is reduced by replacing subsets of components with their prime implicates (unless the number of prime implicates exceeds the number of clauses in the original circuit which can happen for the larger ISCAS circuits).
3. Of the resulting system(s), an exhaustive search is made. With two important modifications: (1) only consider those observations whose number of incorrect outputs is greater than the current best estimate of $MaxCard$, and (2) cache the ATMS conflicts of prior observations.

Table 3 presents the performance compared to those reported in [Feldman, Provan, & van Gemund, 2007].

The algorithm presented in [Feldman, Provan, & van Gemund, 2007] is based on a GDE-like LTMS-based algorithm. The algorithm of this paper is a version of GDE using a conventional ATMS. Initial indications are that GDE benefits significantly from the caching capabilities of the ATMS.

The current algorithm based on [Forbus & de Kleer, 1992] constructs all minimal conflicts and then constructs the minimal diagnoses from the minimal conflicts. This approach constructs far more conflicts than are necessary to find one minimal diagnosis. If there are a large number of minimal diagnoses, most of this is useless work. C6288 can never be analyzed by this approach as it yields an exponential number of minimal conflicts. This result from its inherently parallel structure illustrated in Figure 5.

10 Constructing the MFMC Observation

Once $MaxCard(SD)$ has been identified, the full observation is constructed by inverting the simplifications described earlier. This is always very fast.

One reason MFMC observations are difficult to find is that there are not that many of them. Table 4 lists the number of observation vectors for each minimal cardinality. For the 74182 only 3 % are MFMC observations.

11 Conclusions and Future Work

The approach in this paper has been completely implemented and built upon the Common Lisp code described in [Forbus & de Kleer, 1992]. It outperforms previous approaches and we expect further improvements when implemented with a more modern efficient model-based diagnosis algorithm which does not compute all minimal diagnoses. As this task is easily parallelizable (e.g., using MapReduce [Dean & Ghemawat, 2004]) we plan to use a cluster for future results.

Combining the ATMS-based GDE with the stochastic search could enable finding high cardinality observation vectors for large circuits relatively quickly. Such high cardinality observation vectors are important has they provide high fault coverage for ATPG applications.

12 Acknowledgements

Many conversations with Alex Feldman helped clarify many of the concepts.

Figure 5: The most difficult to analyze circuit from the ISCAS-85 test suite. It is a 16 by 16 bit parallel multiplier built out of half and full adders.

Table 4: Distribution of observation vectors for each minimal diagnosis cardinality for 74182. 480 are MFMC observation vectors.

<table>
<thead>
<tr>
<th>cardinality</th>
<th>MFMC count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>512</td>
</tr>
<tr>
<td>1</td>
<td>2592</td>
</tr>
<tr>
<td>2</td>
<td>5184</td>
</tr>
<tr>
<td>3</td>
<td>5120</td>
</tr>
<tr>
<td>4</td>
<td>2496</td>
</tr>
<tr>
<td>5</td>
<td>480</td>
</tr>
<tr>
<td>Total</td>
<td>16384</td>
</tr>
</tbody>
</table>
References


A Query-Based Approach for Test Selection in Diagnosis - Operating System Discovery as a Case Study

François Gagnon∗ and Babak Esfandiari
Carleton University
{fgagnon,babak}@sce.carleton.ca

Abstract

Test selection in diagnosis is a procedure suggesting tests to be executed when trying to answer the query “What is the diagnosis for this problem?” However, other queries, such as “Is h the diagnosis for this problem?”, are relevant as they can involve faster test selection algorithms and they can result in a lower test execution cost. Usually, a one step lookahead entropy minimization strategy is adopted to implement the test selection procedure. However, we show that this strategy can be arbitrarily bad and therefore, it is important to consider several strategies to solve a query. Each strategy taking a different position in the tradeoff computation time vs test execution cost. In this paper, we consider a query-based approach where test selection is justified and driven by a user’s specific query. We also study different strategies, optimal and approximate, for test selection. Finally, we illustrate how the operating system discovery (OSD) problem can be solved using a diagnosis framework and how it benefits from a query-based approach.

1 Introduction

In [Gagnon et al., 2007], we developed a new approach to operating system discovery (OSD), i.e., finding which operating system is running on a remote computer by analyzing its communication behavior. It soon became clear that our approach can be modeled as a diagnosis system. More precisely, candidate generation helped us in developing the passive module, i.e., computing the set of possible OS based on available communication sessions. However, candidate elimination was not as helpful when it came to developing the active module, i.e., probing the computer to generate specific communication sessions in order to eliminate some OS. Candidate elimination has simply received far less attention than candidate generation. Candidate elimination uses a test selection procedure suggesting tests that will provide the missing information to answer the traditional query: “What is the actual diagnosis?”. However, the focus on that single query is inconvenient since other queries, such as “Is h the actual diagnosis?”, are relevant. Some queries support faster test selection algorithms and others can be solved at a much lower test execution cost.

It has been claimed, e.g., [de Kleer and Williams, 1987], that the computation time required to minimize the number of executed tests is prohibitive compared to the cost of executing a few extra tests. As a consequence, the only strategy for test selection discussed in the diagnosis literature is based on a one step lookahead entropy minimization. Obviously, this strategy can lead to executing more tests than necessary. However, it was not clear whether the entropy minimization strategy guarantees to provide a solution close to the optimal solution, i.e., the one using a minimal number of tests, or if it can be arbitrarily bad. This paper provides a proof that the one step lookahead entropy minimization strategy can be arbitrarily bad. We believe that minimizing the number of tests is an important factor in some domains, e.g., in medicine where tests are very costly (both in terms of money and time required to obtain the results). Our approach is thus to provide the user with a spectrum of algorithms, each focusing on different resources (e.g., computation time or test execution cost), and give him the freedom to select which strategy should be used to solve his query.

This paper presents a query-based approach to diagnosis in which the use of candidate elimination is driven by a specific user’s query (e.g., “What is the actual diagnosis?” or “Is h the actual diagnosis?”). The basic idea is that some queries will intrinsically require fewer tests than others. Moreover, different test selection algorithms to solve one specific query, i.e., “Is h the actual diagnosis?”, are studied: each algorithm concentrates on different factors (test selection time vs test execution cost).

The paper is structured as follows. Section 2 introduces OSD discovery and explains how it can be seen as a diagnosis task. Section 3 introduces the query-based approach and presents some particularly relevant queries. Section 4 details the implementation of candidate generation in our OSD tool. Section 5 presents several test selection strategies to solve one of the queries under different conditions and with different optimization concerns, focusing on the OSD domain.
2 Operating System Discovery as Diagnosis

In this section, we first introduce operating system discovery and discuss the limitations of the classical approaches. Then, we explain how OSD can be modeled as a diagnosis task and how this model helps addressing the limitations of current OSD tools.

2.1 Operating System Discovery

Operating system discovery addresses the problem of finding out which OS is running on networked computers by analyzing their communication behavior. This can be done by exploiting the fact that communication protocols are ambiguously specified, leaving to OS constructors the task to implement details as they see fit. For instance, the Windows family operating systems enable the don’t fragment (DF) bit in their TCP SYN packets while the OS in the Net BSD family disable it. In the same way, other fields allow to distinguish between operating systems of the same family. Below, we introduce the passive and active approaches to operating system discovery and discuss their limitations.

Passive OS Discovery

In passive OS discovery, one only listens on the network and reasons from the packets available. From the (partial) information gathered, one has to guess the OS running on the machine.

The main problem with this approach is that information may not be available when needed as observations cannot be generated on demand. From a diagnosis point of view, this corresponds to a diagnosis system without any tests (sensing actions). Also, in passive OS discovery, only packets from valid communication sequences will be gathered. Usually, more information can be deduced from carefully engineered (and possibly abnormal) stimulus-response sequences. Although passive tools should be monitoring the network and updating their knowledge base on a continuous basis, they turn out to be memoryless and simply analyze each packet individually, i.e., regardless of any other information that could have been known beforehand. Thus, for each packet they provide a guess or two for the operating system (not the set of all possible OS). In diagnosis terms, this corresponds to guessing the actual diagnosis for each observation, disregarding previous observations.

Active OS Discovery

In active OSD, one can directly probe a machine to deduce its operating system based on the reaction of the target to the synthesized stimuli.

The main problem with active OS discovery is the large amount of traffic generated in order to discover the OS. There are several factors why active tools are too noisy. First, active tools usually execute all available tests, regardless of whether a test is relevant or not in this specific situation. Moreover, active tools do not take advantage of packets freely available on the network; they only analyze the packets generated by their own tests. Furthermore, active tools are designed to find out the actual OS running on a machine (i.e., finding the actual diagnosis). Thus if we simply want to know if the OS is Windows 2000 Sp1, we still have to execute all the tests. Finally, active tools often generate abnormal traffic which can interfere with other network components, e.g., an intrusion detection system.

2.2 Diagnosis

Inspired from [Reiter, 1987], we consider a diagnosis system to be a quadruple \((\text{CONST}, \text{OBS}, \text{SD}, \text{TEST})\) where:

- **CONST**: is the set of explanatory constituents available to build a diagnosis (corresponds to COMP in [Reiter, 1987]). See Section 2.3.
- **OBS**: is the set of observations that can possibly be made on the system. See Section 2.4.
- **SD**: is the system description or, more generally, the description of the system behavior. In our case, we consider a rule-based representation of the behavior so SD contains rules associating causes (explanatory constituents) and symptoms (observations). See Section 2.5.
- **TEST**: is the set of tests that can be used to generate observations. See Section 2.7.

Below, we consider individually each of the above components for operating system discovery.

2.3 Explanatory Constituents (**CONST**)

For operating system discovery, **CONST** is the set of possible operating systems. By analogy with medical diagnosis, we will say that the disease (operating system) of a specific patient (computer) is, for instance, chicken pox (Windows 2000 Sp1). Moreover, we consider only single-fault diagnosis. As a consequence, our hypothesis space, from which we select possible diagnosis, is \(\mathcal{H} = \text{CONST}\). A diagnosis candidate \(\Delta\) is thus a single element of \(\text{CONST}\), i.e., a single OS. We can interpret \(\Delta = c\) as the conjecture that the computer is running the OS represented by \(c\) and, obviously, it is not running any of the other operating systems.

There is one case where it would be interesting to consider multiple-fault diagnosis for OSD: when several computers are hidden behind a network address translator (NAT). In such a case, the traffic coming from those machines will appear to come from the NAT, but it will represent different OS behaviors (for the different hidden computers). This NAT situation will pose a problem to our single-fault model (no hypothesis can explain the observations generated by a NAT), but could be addressed nicely with a multiple-fault model. Nevertheless, we decided to use a single-fault model since we want to end up with an empty set of diagnosis candidates when a single OS is not sufficient to explain the observations. This happens, for instance, when we gather observations from an unknown OS or when a user changes the OS of his computer (or with a NAT). Moreover, NAT are not that common in networks and they pose a problem for active OSD (testing) anyway.

2.4 Observations (**OBS**)

Observations in OSD are network events. For simplicity, we consider here that an observation is a packet. But it could also be a more abstract network event such as 3 ARP requests with a delay of 6 seconds in between or a stimulus-response pair of packets (e.g., TCP SYN and TCP SYN/ACK). In practice, a network event never contains more than a few packets.
2.5 System Description (SD)
As mentioned above, we use a rule-based approach for OSD. This is quite natural, since we could hardly build a model of the underlying system. It is not even clear what is that system. One of the main criticisms against rule-based diagnosis is its close relationship to expert systems in which the rules are provided by experts in a very ad hoc manner. However, as it will be discussed in Section 4.2, OSD does not suffer from this drawback. Rules will be generated in a mechanical way from the result of rigorous experiments that can be considered complete.

The rules composing SD will have the form
\[ c_1 \lor c_2 \lor \ldots \lor c_n \leftarrow o \]  
(1)
where \( c_i \in \text{CONST} \) and \( o \in \text{OBS} \). That is, for each observation (network event), we have the complete set of possible causes (operating system) that can explain it.

\[ o \leftarrow c \]  
(2)
We could have used the rule format advocated by [Poole et al., 1987], see (2); however, we believe rules like (1) to be better suited for our task. We based on decision on the following arguments:

- The meaning of (1) is more intuitive than the meaning of (2). (1) means that if \( o \) is observed, then at least one of the \( c_i \) in the consequent must be true, i.e., one of the \( c_i \) is the cause explaining the observation. This is exactly the intended meaning of such a rule, assuming we know everything (i.e., we know every possible cause for observation \( o \)). This assumption is made in the diagnosis process anyway, thus (1) simply make the assumption explicit. On the other hand, (2) logically means that whenever \( c \) is true (i.e., the patient has disease \( c \)), then we observe symptoms \( o \). Unfortunately, this is not the intended meaning of such a rule. The intended meaning is more something like \( c \) “could” cause observation \( o \), but the notion of “could” is not captured by the logical implication. To circumvent this semantics problem, rules like (2) have to be used in an abductive reasoning mechanism.

- There is a problem with unanticipated observations and (2). As mentioned in [Poole, 1985], if we obtain an observation that does not appear in any rule of SD when using rules like (2) and abductive reasoning, then the set of diagnosis candidates is empty. We have to avoid this problem in OSD, since the space of observation is quite large and we expect to gather unanticipated observations. With (1), unanticipated observations pose no problems; they simply cannot be used to refute any hypothesis.

- Encoding our knowledge with rules like (1) will allow the use of very simple, intuitive, and fast algorithms to compute the set of diagnosis candidates for OSD. This will be discussed in Section 4.1.

2.6 Diagnosis Candidate
Based on the previous discussions, we can now provide a definition for diagnosis candidates.

Definition 2.1 (Diagnosis Candidate). Given a set of observations \( \theta \), \( \Delta \in \mathcal{H} \) (recall that \( \mathcal{H} = \text{CONST} \)) is a diagnosis candidate for \( \theta \) iff
\[ SD \cup \theta \cup \{ \Delta \} \cup \{ \neg c | c \in \text{CONST} \setminus \{ \Delta \} \} \]
is consistent.

From there, we can easily define the set of diagnosis candidates for observations \( \theta \), noted \( \Gamma_{\theta} \), as:
\[ \Gamma_{\theta} = \{ \Delta \in \mathcal{H}| \Delta \text{ is a diagnosis candidate for } \theta \} \]
The use of consistency-based reasoning is justified by the form of the rules in SD (1), see the discussion in [Poole, 1994].

2.7 Tests (TEST)
A test in OS discovery consists of sending a stimulus to a computer and analyzing the response. Tests can always be executed, thus they do not have any pre/post conditions. Moreover, there are no state altering actions in OSD, thus we consider only tests and not actions in general.

From a diagnosis point of view, a test \( t \) is represented by a prediction function \( P_t : \mathcal{H} \rightarrow \wp(\text{OBS}) \). That is, given a hypothesis \( h \), we know the observations that will be gathered if we were to execute the test \( t \) in a situation were \( h \) is the actual diagnosis.

For instance, one OSD test consists of sending a TCP SYN packet on a closed port of the target computer and analyzing the TCP RST/ACK packet it will produce as a response. We call this test the rst test and its prediction function is partially given here:
\[ P_{\text{rst}}(\text{Windows 2000 Sp1}) = \{ \text{tcp(yes, rstack, 255)} \} \]
\[ P_{\text{rst}}(\text{MacOS 10.1.4}) = \{ \text{tcp(no, rstack, 64)} \} \]
\[ \text{etc.} \]
This basically means that if we send a SYN packet on a closed port of a computer running Windows 2000 Sp1, then it will respond with a RST/ACK packet in which the don’t fragment (DF) bit is enabled and with a time-to-live (TTL) of 255. This is different from the behavior of MacOS, which would disable the DF bit and use a TTL of 64.

3 Query-Based Diagnosis
In Section 2.1, we saw two approaches to OS discovery: active and passive. The main idea of using diagnosis to solve the OSD problem is to unify these two approaches into a hybrid tool that will not suffer from the drawbacks of the existing tools, see [Gagnon et al., 2007].

To do so, we propose a query-based approach to diagnosis, see Figure 1, where test selection is driven by a user’s query. More precisely, every time \( n \) observations are made (step 1 in Figure 1), the candidate generation process is launched to update the current set of diagnosis candidates (steps 2 and 3). Thus the information conveyed by observations (network packets in our case) analyzed earlier is conserved by the set of diagnosis candidates, while the observations themselves are

\[ ^1 \text{We impose the following restriction on the prediction function of a test } t: h \in \Gamma_{P_t(h)}. \]
We could answer the new queries by answering the classical query and then interpreting the result in terms of the other queries. For instance, to answer the single-candidate query, one can first answer the classical query and compare the actual diagnosis $\Delta$ with $h$. Moreover, the single-candidate query with $h$ is simply a special case of the multiple-candidate query with $H$ where $H = \{h\}$. We distinguish between these two queries because we believe the single-candidate query to be computationally easier to solve optimally (by using as few tests as possible) than the multiple-candidate query in some cases (notably in OSD). The idea behind the new queries is that they can sometimes (hopefully often) be solved by executing fewer (hopefully much fewer) tests than the classical query, see Proposition 3.1 and Example 3.1 below.

**Proposition 3.1.** To know if $h \in \mathcal{H}$ is the actual diagnosis, solving the single-candidate query with $h$ requires at most as many tests as solving the classical query.

**Proof:**

The same set of tests used to solve the classical query can be used to solve the single-candidate query.

**Example 3.1.** Consider the $n$ tests $t_1, t_2, \ldots, t_n$ and the $n$ hypotheses $h_1, h_2, \ldots, h_n$. Assume each test $t_i$ has two possible outcomes:

- refutes only $h_i$, or
- confirms only $h_i$, i.e., refutes every hypothesis except $h_i$.

Starting from $\Gamma = \{h_1, h_2, \ldots, h_n\}$, we can solve the single-candidate query for any $h_i$ with a single test, namely $t_i$. On the other hand, we need $n - 1$ tests to be guaranteed to solve the classical query. Thus, solving the classical query can require drastically more tests than solving the single-candidate query.

### 4 Candidate Generation

Here, we present an algorithmic solution to the candidate generation problem for the case of operating system discovery, Section 4.1. We also explain, in Section 4.2, how the rules of SD are automatically generated, thus avoiding the need for an expert to create and update SD.

#### 4.1 Computing Candidates

Let's consider an alternative syntactic form for the rules in SD. Instead of the purely logical form used in (1), we use the following form:

$$\{c_1, c_2, \ldots, c_n\} \leftarrow o$$

(3)

conserving the same meaning. We also assume, as it is the case in OSD, that SD is such that any given observation $o \in \text{OBS}$ appears in at most one rule. Finally, recall that we are looking for single-fault explanations only.

Note that with the rule format (3), $\{c_1, c_2, \ldots, c_n\}$ is one conflict set $^3\text{ of } \theta \subseteq \text{OBS} \text{ iff } o \in \theta$. This will be the basis of our candidate generation algorithm.

^3A conflict set for $\theta \subseteq \text{OBS}$ is $C \subseteq \text{CONST}$ such that $\text{SD} \cup \theta \cup \{-c: c \in C\}$ is inconsistent, see [Reiter, 1987].
CanGen(\(SD, \text{CONST}, \Gamma, \theta\))

Provides the diagnosis candidates for \(\theta\)

Input: \(SD\): the set of rules \(\{c_1, c_2, \ldots, c_n\} \leftarrow o^*\)
\(\text{CONST}\): the set of explanatory constituents
\(\Gamma\): the current set of diagnosis candidates sorted
\(\theta\): the set of observations to explain

Output: \(\theta\) the set of diagnosis candidates

1. \(C \leftarrow \text{conflict sets of } SD \text{ and } \theta\) \(O(|\theta| \log |\theta| + |SD|)\)
2. FORALL \(C \in C\) \(O(|\theta|)\)
2-1. \(\Gamma \leftarrow \Gamma \cap C\) \(O(|\theta| \times |\text{CONST}|)\)
3. RETURN \(\Gamma\) \(O(1)\)

Figure 2: Candidates Generation Algorithm

With this in mind, we can now provide an algorithm to compute the set of diagnosis candidates for the observations \(\theta\) \(\subseteq\) \(OBS\). The algorithm is shown in Figure 2 and explained below.

The algorithm works as follows. First, compute the collection \(C\) of conflict sets of \(\theta\) (step 1). Then compute the minimal hitting sets\(^3\) of \(C\) (steps 2. and 2-1.). As mentioned in [Reiter, 1987], each minimal hitting set is a minimal diagnosis candidate. But since we are only interested in single-fault explanations, those minimal hitting sets will be singletons. Now if \(S\) is a singleton hitting set of \(C\), it means that \(S \in \cap_{C_i \in C} C_i\). The converse is also true, any element of \(\cap_{C_i \in C} C_i\) is a singleton hitting set of \(C\). Thus the intersection of all our conflict sets will provide the diagnosis candidates. Since we are only interested about the candidates that were previously possible diagnosis candidates, we restrict our search to \(\Gamma\), the previous set of diagnosis candidates (initially, \(\Gamma = \emptyset\)).

Before getting into the analysis of this algorithm, consider the following. The important parameters for the time complexity will be \(|SD|\), \(|\text{CONST}|\) and \(|\theta|\). In a specific tool, only \(|\theta|\) will vary from one run to the other\(^6\). To have an idea of the size of those parameters, in our current OSD tool, we have 433 rules in \(SD\) (\(|SD| = 433\)) and 208 different OS \((|\text{CONST}| = 208)\). Note that we consider different release of an OS to be distinct, e.g., Windows 2000 Sp1 is different from Windows 2000 Sp4. Note also that we can control the size of \(\theta\) by triggering the candidate generation procedure every time \(|\theta|\) reaches a threshold. In our tool, we use 100 for that threshold. After a run of candidate generation, the observations are discarded; only the diagnosis information they convey is kept by remembering the resulting set of diagnosis candidates (and using it for the next run of candidate generation). The analysis of the algorithm is as follows:

1. We first compute the collection of conflict sets for \(\theta\). This can be done in \(O(|\theta| \log |\theta| + |SD| + |\theta|)\) in the following way. Consider the rules of \(SD\) to be sorted according to their antecedent, with some complete order over the observations. This sorting can be done once, off line, and thus is not considered in the computation time. Start by sorting the observations in \(\theta\) according to that same order, this requires \(O(|\theta| \log |\theta|)\). Now we only need to traverse \(SD\) and \(\theta\) once, simultaneously to see which rules are triggered by \(\theta\) and gather the conflict sets, this requires \(O(|SD| + |\theta|)\).

2. We consider every conflict set individually. Since each observation in \(\theta\) can contribute to at most one conflict set\(^5\), there are at most \(|\theta|\) conflict sets. We also have at most \(|SD|\) conflict sets, but \(|\theta|\) is considered smaller than \(|SD|\).

2-1. We keep only the diagnosis candidates of \(\Gamma\) that explain every observation in \(\theta\), i.e., those that cover every hitting set. This is done by intersecting \(\Gamma\) with each conflict set. Since the intersection is on two sets of size at most \(|\text{CONST}|\), it can be done in \(O(|\text{CONST}|)\). If we assume that both \(\Gamma\) and \(C\) (a conflict set) are sorted, then the intersection can be done in \(O(|\text{CONST}|)\). Each rule of \(SD\) can be such that its consequent (the conflict set to be sorted) is done off line when \(SD\) is created and thus does not contribute to the complexity of the candidate generation algorithm. The fact that \(\Gamma\) is also sorted is discussed below.

3. Finally, we return \(\Gamma\), the set of (single-fault) diagnosis candidates which might be possibly empty (if it is not possible that a single OS generated all the observations). \(\Gamma\) also has to be sorted, since the \(\Gamma\) returned here will be used as the input of the next call to the candidate generation procedure. Assume \(\Gamma\) is sorted when calling candidate generation, it is easy to keep its updated version (updated by intersection) sorted as well. Thus the \(\Gamma\) we return at after computing the candidates is sorted. Initially, \(\Gamma = \emptyset\) and we simply have to pre sort this particular set.

The final analysis of the candidate generation above is: \(O(|\theta| \log |\theta| + |SD| + |\theta| \times |\text{CONST}|)\), a very reasonable result for a practical application.

4.2 Automatic Generation of SD

One of the difficulties of rule-based diagnosis is to come up with the rules. Traditionally, the rules are provided by an expert in an ad hoc manner. Since the items of interest for us are operating systems, we can consider the possibility of installing every available OS on distinct machines and studying their behavior in a systematic way. However, such an approach would be expensive due to the physical resources (several computers) and the tedious operations (manipulating each computer to generate traffic) required.

To circumvent this problem, we created a virtual network experiment controller in which we can specify experiments that will be executed in a virtual environment using VMWare virtual machines. The specification language allows us to both select the virtual machines (VM) involved in an experiment and the actions to be performed by each of them.

With this environment, we can for instance tell the controller to open two VM, a traffic recorder and the studied OS. Then the studied OS is to perform some actions (e.g., open a web browser and a FTP connection to generate TCP SYN

---

\(^{3}\)A hitting set for a collection of sets \(C\) is a set \(S \subseteq \bigcup_{C_i \in C} C_i\) such that \(S \cap C_i \neq \emptyset\) for every \(C_i \in C\).

\(^{4}\)However, in the case of OSD, \(|SD|\) and \(|\text{CONST}|\) can vary from one version of the tool to the other, as new OS are released every year.

\(^{5}\)Recall that each \(o \in OBS\) appears in at most one rule.
packets). The traffic is recorded and later analyzed. From this traffic, it is easy to see what is the behavior of an OS (e.g., does it enable the DF bit or not). Finally, the rules in SD are easily generated from this analysis.

This process is fully automated, except for the creation of the virtual machines. Once a new operating system is released, we simply need to build a new VM with this OS and launch the experiment to analyze its behavior and update SD.

5 Candidate Elimination - Test Selection

The literature on test selection in diagnosis is divided into two trends:

- the specification of tests, see [McIlraith, 1994], from which our specification is inspired, and [Struss, 2005].
- the use of an entropy minimization heuristic to find the actual diagnosis, see [de Kleer and Williams, 1987].

We believe that a broader spectrum of strategies to solve the queries should be studied. In that perspective, we study several strategies to solve the single-candidate queries.

In this section we address the problem of test selection: which tests should be performed next to allow us to answer a query. We focus on the single-candidate query case, but we discuss different strategies for test selection (minimizing the number of executed tests, minimizing the test selection effort) under different conditions (cost of tests, prior probabilities). But first, we explain how to reason about the outcome of tests.

5.1 Test Outcome Interpretation

Given our definition of tests, we can predict the observations generated by a test for a specific hypothesis. However, this is of limited use for reasoning on the impact of executing a specific test, as observations need to be interpreted to provide possible candidates. Here we discuss the interpretation of a test to provide the necessary tools for reasoning.

Definition 5.1 (Interpretation). The interpretation of a test outcome \( P_i(h) \) is \( \Gamma_{P_i(h)} \), i.e., the set of diagnosis candidates explaining the observations \( P_i(h) \).

Note that the notion of interpretation depends on the reasoning mode to compute diagnosis candidates. However, unlike [McIlraith and Reiter, 1992], it provides a level of abstraction such that test selection algorithms are not affected by changing the reasoning mode (abduction, consistency-based or both [Console and Torasso, 1991]).

Informally, we consider a state as being a set of diagnosis candidates. The current state is the set of diagnosis candidates we currently consider. The initial state in OSD is \( \mathcal{H} \), i.e., the set of all hypotheses.

Definition 5.2 (Possible Interpreted Outcomes). The set of possible interpreted outcomes of a test \( t \), denoted \( \text{PIO}(t, \Gamma) \) is the set of states we could end up in after executing \( t \) in state \( \Gamma \). More formally,

\[
\text{PIO}(t, \Gamma) = \{ \Gamma \cap \Gamma_{P_i(h)} | h \in \Gamma \} \tag{4}
\]

With the notion of possible interpreted outcomes of a test, we can now reason about performing a test in a given state and thus we are in a good position to select tests with respect to different preference criteria and/or to build plans (sequences or branching sequences of tests) in order to achieve a goal (i.e., answer a query).

5.2 Single-Candidate Query

Here we provide algorithmic solutions to answer the single-candidate query for operating system discovery. In OSD, the tests are uniquely supporting, see Definition 5.3 below.

Definition 5.3 (Uniquely Supporting Test). A test \( t \) uniquely supports a hypothesis \( h \) if it has only one outcome \( \Gamma' \in \text{PIO}(t, \Gamma) \) such that \( h \in \Gamma' \), i.e., \( t \) has only one outcome confirming (supporting) \( h \). A test is uniquely supporting if it uniquely supports every hypothesis.

Note that our notion of uniquely supporting test is stronger than the notion of discriminating test from [McIlraith, 1994], in the sense that every uniquely supporting test is also a discriminating test. However, a uniquely supporting test is weaker than an individually discriminating test from [McIlraith, 1994], because after the execution of a uniquely supporting test we may still not know whether a particular hypothesis is true or not.

Note also that a uniquely supporting test can never fail. We can represent tests that may fail by adding the outcome \( \emptyset \) to its set of possible outcomes. Since a possible outcome generates no observations, this particular outcome changes nothing to the set of diagnosis candidates.

The impact of considering only uniquely supporting tests for the single-candidate query for \( h \) is quite interesting: when executing a test \( t \) in state \( \Gamma \), there is only one resulting state \( \Gamma' \) that will still consider \( h \) as a candidate. Thus we only need to focus on \( \Gamma' \), since every other outcome will refute \( h \) (the answer to the query would then be “no”). As a consequence, we can easily characterize the concept of a solution to the single-candidate query.

Definition 5.4 (Solution). Assuming that all tests are uniquely supporting, a solution to the single-candidate query for \( h \) in state \( \Gamma_0 \) is a sequence of tests \( T = [t_1, t_2, \ldots, t_n] \) such that \( \Gamma_n = \{ h \} \) where \( \Gamma_i = \Gamma_{i-1} \cap \Gamma_{P_i(h)} \). That is, after the execution of the tests in \( T \) we are guaranteed to have an answer to the query.

A solution is optimal if there is no solution of lower cost, where the cost of a solution is the sum of the costs of its individual tests.

Given a sequence \( T \) of tests, we can check if \( T \) is a solution to the single-candidate query for \( h \) in state \( \Gamma \) in \( O(|T| \times |\text{CONST}|) \). An intersection over two subsets of \( \text{CONST} \) (this can be done in \( O(|\text{CONST}|) \) if they are sorted) is performed for every test in \( T \).

Minimizing the Number of Executed Tests

Let us first assume that all tests have an equal cost and that we are interested in minimizing the number of executed tests. We can come up with a simple brute force algorithm to find an optimal solution for the single-candidate query: For every subset of \( \text{TEST} \) of increasing size, check if that subset is a solution and if it is, return it as the optimal solution. In the

---

worse case, this will test every subset of TEST to see if it is a solution. This will require, in the worst case, a time in \( O(|\text{CONST}| \times 2^{|\text{TEST}|}) \), see (5).

\[
\sum_{t \in \text{TEST}} |\text{CONST}| \times |T| = |\text{CONST}| \times \sum_{i=0}^{|\text{TEST}|} C_i \text{TEST}_i \tag{5}
\]

Although this algorithm guarantees to return an optimal solution, it may require exponential time in the number of tests available. Thus, the test selection time can be quite high. Next, we consider alternative strategies where the concern is not necessarily about minimizing the number of executed tests.

**Minimizing Test Selection Effort**

Let us now focus on a strategy to minimize the test selection effort. An obvious solution is simply to execute all tests or to randomly select which test to perform until the query can be answered. This strategy requires no test selection effort, as it can be done in constant time. However, it will produce arbitrarily bad solutions compared to the optimal one. Surely, we can have something in between: approximate the optimal solution without too much effort on test selection.

**A Simple Greedy Approximation**

Ideally, we would want a strategy that provides good solutions (not optimal but not arbitrarily bad either) without requiring too much effort on test selection. We can adapt and simplify the entropy minimization strategy, see [De Kleer and Williams, 1987], for the single-candidate query for \( h \) in state \( \Gamma \) with equiprobable hypotheses. Until \( \Gamma \) allows to answer the query (i.e., either \( \Gamma = \{h\} \) or \( h \not\in \Gamma \)), select the test from TEST with the highest discriminant power for \( h \) with respect to \( \Gamma \) (see Definition 5.5 below) and execute it to update \( \Gamma \) with the resulting observations.

**Definition 5.5 (Discriminant Power).** Given a test \( t \), a set of diagnosis candidates \( \Gamma \) and a specific hypothesis \( h \), the **discriminant power** of \( t \) for \( h \) with respect to \( \Gamma \) is the number of candidates from \( \Gamma \) that are eliminated by the outcome of \( t \) which confirms \( h \). That is, \( |\Gamma| - |\Gamma \cap \Gamma_t(h)| \)

That last algorithm requires going through the set of remaining tests each time we need to select the “best” test. As the number of tests to consider decreases as tests are executed, the algorithm as a worst case of \( O(|\text{TEST}|^2) \), if we have to execute every test. Sorting is not useful here, because each time we perform a test the set of diagnosis candidates will change and the discriminant power of a test is relative to the current set of candidates.

Although this algorithm is simple and fast, it can be shown that its solution can be arbitrarily bad. Below is an example where the optimal solution requires only two tests, but we can force the greedy approximation to use as many tests as we want (by doubling the size of the hypothesis space and adding one test).

**Example 5.1.** Consider a situation where \( \Gamma_0 = \{h_0, h_1, \ldots, h_{2k}\} \) and the single candidate query for \( h_0 \) in state \( \Gamma_0 \). Consider also the following \( k+1 \) uniquely supporting tests:

<table>
<thead>
<tr>
<th>Tests</th>
<th>( T_0 )</th>
<th>( T_1 )</th>
<th>( T_{i-1} )</th>
<th>( T_{k-1} )</th>
<th>( T_k )</th>
<th>( T_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>( 2^{k-1} )</td>
<td>( 2^{k-2} )</td>
<td>( 2^{k-3} )</td>
<td>( 2 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>( t_h )</td>
<td>( 2^{k-1} )</td>
<td>( 2^{k-2} )</td>
<td>( 2^{k-3} )</td>
<td>( 2 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>( 2 )</td>
<td>( 2 )</td>
<td>( 2 )</td>
<td>( 2 )</td>
<td>( 2 )</td>
<td>( 2 )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( 4 )</td>
<td>( 4 )</td>
<td>( 4 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( t_{k-1} )</td>
<td>( 2^{k-1} )</td>
<td>( 2^{k-2} )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 0 )</td>
</tr>
</tbody>
</table>

\( t_0: \Gamma_0(h) = \{h_0\} \cup \{h_i\} \) odd

\( t_h: \Gamma_0(h) = \{h_0\} \cup \{h_i\} \) even

\( t_1: \Gamma_0(h) = \{h_0\} \cup \{h_1, h_2\} \cup \{h_5, \ldots, h_{2k}\} \)

\( t_2: \Gamma_0(h) = \{h_0\} \cup \{h_1, \ldots, h_4\} \cup \{h_9, \ldots, h_{2k}\} \)

\( t_{k-1}: \Gamma_0(h) = \{h_0\} \cup \{h_1, \ldots, h_{2k-1}\} \)

The optimal solution is \( \{t_0, t_h\} \) and has size 2. The solution provided by the greedy algorithm would be of size \( k+1 \) (it includes all tests). In state \( \Gamma_0 \), three tests \( \{t_0, t_h, t_{k-1}\} \) have the highest discriminant power (see Table 1); assume the greedy algorithm selects \( t_{k-1} \). In the resulting state, \( \Gamma_1 \) again three tests have the highest discriminant power; assume the greedy algorithm selects \( t_{k-2} \). This process will go on until \( \Gamma_{k} = \{h_0, h_1, h_2\} \) and \( t_0 \) and \( t_h \) are the only tests that have not been executed. The greedy approach will then select successively \( t_0 \) and \( t_h \) to finally provide a solution.

The key idea of Example 5.1 is that increasing the value of \( k \) by one increases the size of the greedy solution by 1, while the optimal solution remains the same. Thus, we can build an instance where the greedy solution is arbitrarily bad.

**Tests with Costs**

Assume now that we consider the tests to have different costs. This is the case in OSD as some tests produce more packets than others and some tests produce abnormal packets. The brute force algorithm discussed previously still works in finding an optimal solution, except we cannot stop as soon as we find a solution, since a solution of higher cardinality might be less costly. The complexity remains the same.

For the greedy approximation, selecting the test with maximal discriminant power or the one with minimal cost would be bad strategies. Selecting the test with the maximal (discriminant power)/cost ratio seems a much better approach. Obviously, the time complexity remains as before. Unfortunately, the solution returned by this algorithm can also be arbitrarily bad, since Example 5.1 is a special case of the new greedy algorithm for tests with costs (where each test has a cost equal to one).

**Prior Probabilities for Hypotheses**

So far, we assumed a uniform distribution of prior probabilities for the hypothesis space. In some cases, we might need

\(^*\)We could easily build an example where \( t_0 \) and \( t_h \) never have the highest discriminant power until they are the only tests left. It would simply be longer and more tedious.
to consider distinct prior probabilities for each hypothesis. It is the case in OSD, since a computer is more likely to be running Windows 2000 Sp4 than Open BSD 1.2.1. Dealing with prior probabilities requires some modifications in our representation. We want $\Gamma : \mathcal{H} \rightarrow [0,1]$ to be a function associating with each hypothesis its probability of being the actual diagnosis. Thus, the candidate generation algorithm needs to update those probabilities. Now the greedy algorithm consists of selecting the test that will minimize the resulting entropy. We have to adapt the notion of entropy to the single-candidate query. The entropy of a state $\Gamma$ is then:

$$E(\Gamma) = \begin{cases} 0 & \text{if } \Gamma(h) = 0 \\ -\sum_{h \in \mathcal{H}} \Gamma(h) \log \Gamma(h) & \text{otherwise} \end{cases}$$

(6)

Although [de Kleer et al., 1992] provided an example where this one-step lookahead strategy performs very well,[[1]]#1 it should not come as a surprise that the solution returned by this greedy algorithm based on entropy minimization can, once again, be arbitrarily bad. Example 5.1 is a special case where prior probabilities of hypotheses are equal.

6 Conclusion & Future Work

In this paper, we first presented how operating system discovery can be seen as a diagnosis task. The resulting tool for OSD definitely has an edge over other existing tools, as the results of experiments in [Gagnon et al., 2007] demonstrate. Second, since the focus on test selection for candidate elimination has always been on finding out what the actual diagnosis is, we proposed a query-based approach to diagnosis. This approach provides more flexibility to the user and can result in significantly fewer tests being executed, when asking easier queries. Finally, we studied different test selection strategies to solve the single candidate query in the context of OSD, i.e., with uniquely supporting tests. Of particular interest is the result that elementary approximation algorithms can provide arbitrarily bad solutions. This is the case, for instance, with the popular entropy minimization algorithm.

The work presented here can be extended in several directions. First, since the approximation algorithms are all arbitrarily bad, it seems important to continue the search for better approximation algorithms, hopefully finding one guaranteed to provide a solution close to the optimal one.

Since we studied test selection only for the single-candidate query with the assumption that tests are uniquely supporting, the next important step is to study test selection for the other queries. This will be a challenge since the notion of optimal solution is not easily defined in those cases. Without a notion of optimal solution, discussing approximation algorithms is not very meaningful, since we have no idea how good (or bad) those approximations are.

The queries we proposed here are from the OSD domain. We believe that other diagnosis domains could provide different queries or different ways of interpreting our queries. For instance, finding the actual diagnosis in a domain with probabilistic hypotheses does not mean refuting every hypothesis but one. The objective there is to decrease the probability of every hypothesis but one below a threshold.

Finally, the notion of tests as defined here, mainly the prediction function, is not general enough. Currently, the outcome of a test is deterministic with respect to a hypothesis, i.e., we know the exact observations that will result from executing a test. But this will not always be the case. Moreover, the prediction function must be defined for every element of $\mathcal{H}$. In the multiple-fault case, this will be tedious. It would be preferable to define it over CONST and to predict the outcome for a multiple-fault hypothesis by composing the prediction for the underlying explanatory constituents. Different composition operators (e.g., union, intersection, etc.) will be required for different domains.

References


---

A Comparison of Two Methods for Fault Detection: a Statistical Decision, and an Interval-based Approach

Esteban R. Gelso†, Gautam Biswas‡, Sandra M. Castillo‡, and Joaquim Armengol‡

†IIiA - Universitat de Girona, Campus de Montilivi, Girona, E-17071, Spain
‡Institute for Software Integrated Systems (ISIS)
Dept. of EECS, Vanderbilt University, Nashville, TN 37237, USA

Abstract

This paper compares two approaches for robust model-based fault detection. The first method is based on the use of an Extended Kalman filter for tracking dynamic system behavior and a Z-test for robust detection on non-zero residual signals. The second method combines interval methods and constraint satisfaction techniques for consistency-based fault detection. Both of them were applied to an example to show the influence of the choice of parameter values on the detection performance.

1 Introduction

Diagnosis can be approached from different perspectives according to the nature of the knowledge available about the system and faults that occur in the system. The so-called model-based diagnosis (MBD) approaches use an explicit model of the system to be diagnosed [Cordier et al., 2004]. Fault occurrences in the sensors, actuators, and the physical process are defined by discrepancies between observed behavior and that predicted by the model.

The goal of diagnosis is to detect faults early, so corrective actions can be taken before the system fails. Early detection of non-zero residual signals is complicated by the presence of noise in the measured data, inaccuracy of the sensors, disturbances to the system, and modeling errors.

This is especially true because one has to avoid missed alarms and false positives (alarms) in diagnosis tasks that are directed to real world applications. A “false alarm” occurs when the fault detection system returns a wrong result, i.e., it indicates a fault while in reality no fault has occurred. A “missed alarm” occurs when there is a fault but it can not be detected. In all the methods, there is a trade-off between the false alarm rate, which may be attributed to overly sensitive detection, and the missed alarm rate or detection delay, which may be attributed to lack of sufficient sensitivity in the detection mechanism. Too many false alarms make the system operation unreliable, but trying to reduce the number of false alarms by increasing the detection threshold typically results in an increased probability of missed detection [Chowdhury et al., 2006].

Different approaches for fault detection using mathematical models have been developed in over the last two decades.

These methods are based, e.g., on parameter estimation, parity equations or state observers. A general survey of supervision, fault detection and diagnosis methods is given in [Isermann, 1997; 2005].

Accuracy of the dynamic model plays a big role in determining the reliability of the detection scheme. However, modeling errors and disturbances in complex engineering systems are unavoidable and, hence there is a need to develop robust fault diagnosis algorithms. The goal of robustness is to minimize the false and missing alarm rates due to the effects that modeling uncertainty and unknown disturbances will have on the residuals. This can be achieved in several ways, e.g., by statistical data processing, averaging, or by finding and using the most effective threshold.

This paper compares two approaches for robust fault detection. The first one (see Fig. 1) is based on (i) a robust observer to track the nominal system dynamics, and (ii) a fault detector which monitors the difference in the observed and expected behavior (residual) using a statistical testing method. The observer is implemented as an Extended Kalman filter [Gelb, 1996]. Model uncertainty and measurement noise are implemented as white, uncorrelated Gaussian distributions with zero mean. The second approach (see Fig. 2) represents uncertainties in the system model and measurements by an interval model, i.e., a model in which the parameters take interval values. The fault detection problem is represented as a constraint satisfaction problem and the resolution of this problem can be performed by combining interval methods and constraint satisfaction techniques.

The paper is organized as follows. Section 2 presents the two approaches for robust fault detection. In Section 3, an application example is described, and fault detection results are presented. Finally, some conclusions are given at the end of the paper in Section 4.
2 Fault detection

In this section, two methods for robust fault detection are presented and main differences between them are noted. Both approaches assume that the system dynamics can be modeled in the discrete-time nonlinear form:

\[
\begin{align*}
\mathbf{x}(k+1) &= g(\mathbf{x}(k), \mathbf{u}(k), \theta) + \mathbf{w}(k) \\
\mathbf{y}(k) &= h(\mathbf{x}(k), \mathbf{u}(k), \theta) + \mathbf{v}(k)
\end{align*}
\]

where:

\( \mathbf{u}(k) \in \mathbb{R}^{n_u} \), \( \mathbf{y}(k) \in \mathbb{R}^{n_y} \), and \( \mathbf{x}(k) \in \mathbb{R}^{n_x} \) are the input, output, and state vector, respectively.

\( \mathbf{w}(k) \in \mathbb{R}^{n_w} \) and \( \mathbf{v}(k) \in \mathbb{R}^{n_v} \) are noise or perturbation vectors, as they are described for each method (see Section 2.1 and 2.2).

\( \theta \in \mathbb{R}^{n_\theta} \) is a vector of the model parameters that describe system behavior.

\( \mathbf{x}(k) \) is the state at the time step \( k \) + 1 to the current time step \( k \), and the non-linear function \( h \) relates the state \( \mathbf{x}(k) \) to the measurement \( \mathbf{y}(k) \).

2.1 Extended Kalman Filter and Z-test

Fault detection, as performed in the TRANSCEND system [Mosterman and Biswas, 1999; Biswas et al., 2003; Roychowdhury et al., 2006], includes two modules: (i) the observer, and (ii) the fault detector.

The observer, implemented as an Extended Kalman filter (EKF) [Gelb, 1965], takes as input the signals and sensor measurements, and estimates the state \( \mathbf{x} \) as well as outputs, of a discrete-time system that is governed by a nonlinear stochastic difference equation such as Eq. 1.

The random variables \( \mathbf{w}(k) \) and \( \mathbf{v}(k) \) represent the process modeling errors and measurement noise, respectively. They are assumed to be independent, white, and with normal probability distribution \( p(\mathbf{w}) \sim N(0, Q) \) and \( p(\mathbf{v}) \sim N(0, R) \), where the process noise covariance \( Q \) and measurement covariance \( R \) matrices are assumed to be constant. The process covariance matrix captures the effect of modeling errors and unknown input disturbances to the system. The measurement covariance matrix models sensor discrepancies and measurement noise.

Because the EKF linearizes estimation around the current time point of the tracking and estimation process, the results obtained are solutions of the linear approximation and are expected to be an approximation of the solution of the nonlinear problem. This approximation may not work well in the presence of strong nonlinearities.

The fault detector monitors the measurement residual, \( r(k) = \hat{y}(k) - \tilde{y}(k) \), at every time step, where \( \hat{y} \) is the measured value, and \( \tilde{y} \) is the expected system output, determined by the observer. Ideally, any non-zero residual value implies a fault, which should trigger the fault isolation scheme. In most real systems, the measured values are corrupted by noise, and the system model (thus the prediction system) is not perfect. Therefore, statistical techniques are required for reliable fault detection.

The fault detector uses a sliding window scheme to compute the residual signal value at time step \( k \), i.e.,

\[
\hat{\mu}(k) = \frac{1}{N_2} \sum_{i=k-N_2+1}^{k} r(i),
\]

where \( N_2 \) is the predefined window size.

A hypothesis testing scheme based on the Z-test is employed to establish the significance of the deviation. To perform the Z-test, the variance of the measurement residual must be known. We estimate the variance of the signal using Eq. 3, but from a larger data sample of size \( N_1 \), i.e., \( N_1 >> N_2 \) (Fig. 3). The considered \( \text{VarDelay} \) guarantees that the variance is computed using data from the system under normal working conditions.

\[
\sigma^2_N(k) = \frac{1}{N_1-1} \sum_{i=k-N_1+1}^{k} (r(i) - \mu_N(k))^2
\]

We assume the variance of the signal remains unchanged after fault occurrence.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
The Z-test is employed in the following manner:

$$\mu_- < \hat{\mu}_N < \mu_+ \Rightarrow \text{No Fault}$$

otherwise \( \Rightarrow \text{Fault} \)

To accommodate measurement noise, inaccuracies in the model and sensitivity of the detection scheme one has to trade-off false alarm generation versus detection delays. Statistical hypothesis testing schemes help to reduce the false alarm rate, but introduce a delay between the time of occurrence and detection of faults [Roychoudhury et al., 2006].

### 2.2 Interval-based Consistency Technique

Another way to represent the uncertainties in the model and measurements is by means of an interval scheme, where the parameters and measurements take on values within defined intervals. In some situations, we can say about the values of a variable, \( x \) is bounded by values \( [\underline{x}, \bar{x}] \), \( \underline{x} \leq x \leq \bar{x} \) with certainty but the distribution within this interval may be unknown [Persson et al., 2007]. This situation occurs often in measurement practice when, for a sensor, the only information we may have are the bounds on its error values.

Many engineering problems (e.g., parameter and state estimation, robust control design problems) can be formulated in a logical form by means of some kind of first order predicate formulas: formulas with the logical quantifiers (universal and existential), a set of real continuous functions (equalities and inequalities), and variables ranging over real interval domains.

As defined in [Shary, 2002], a numerical constraint satisfaction problem is a triple CSP = \( (\mathcal{V}, \mathcal{D}, \mathcal{C}(x)) \) defined by

1. a set of numeric variables \( \mathcal{V} = \{x_1, \ldots, x_n\} \),
2. a set of domains \( \mathcal{D} = \{D_1, \ldots, D_n\} \) where \( D_i \), a set of numeric values, is the domain associated with the variable \( x_i \),
3. a set of constraints \( \mathcal{C}(x) = \{C_1(x), \ldots, C_m(x)\} \) where a constraint \( C_i(x) \) is determined by a numeric relation (equation, inequality, inclusion, etc.) linking a set of variables under consideration.

The fault detection problem can be represented by a CSP similar to the one presented in [Jaulin, 2002], which deals with the problem of nonlinear state estimation. In other words, the task is to solve for the values of the state variables, \( \mathbf{x}(k) \), \( 1 \leq k \leq n \) given \( \mathbf{x}(0) \), input \( \mathbf{u}(k) \), \( 1 \leq k \leq n \), and output \( \mathbf{y}(k) \), \( 1 \leq k \leq n \).

Starting with a discrete-time nonlinear dynamic system like the one described by Eq. 1, where:

\[ \mathbf{y}(k) = \mathbf{h}(\mathbf{x}(k), \mathbf{u}(k), \theta, \mathbf{w}(k)) \]

\[ \mathbf{w}(k) \in \mathbb{R}^m \] and \( \mathbf{v}(k) \in \mathbb{R}^m \) are the perturbation and measurement noise vectors, which are unknown but bounded. The perturbation vector takes into account, for instance, unmodeled dynamics of the actual plant, unknown inputs, or an error due to the discretization procedure.

\( \theta \in \mathbb{R}^n \) is a vector of interval bounded parameters, where the parameters values can be considered as time variant or invariant, but always within fixed bounds.

The CSP corresponding to the dynamic system can be represented as:

\[ \mathcal{V} = \{\theta, \tilde{\mathbf{y}}(1), \ldots, \tilde{\mathbf{y}}(k), \tilde{\mathbf{x}}(1), \ldots, \tilde{\mathbf{x}}(k+1), \tilde{\mathbf{u}}(1), \ldots, \tilde{\mathbf{u}}(k) \} \]

\[ \mathcal{D} = \{\Theta, \tilde{\mathbf{Y}}(1), \ldots, \tilde{\mathbf{Y}}(k), \tilde{\mathbf{X}}(1), \ldots, \tilde{\mathbf{X}}(k+1), \tilde{\mathbf{U}}(1), \ldots, \tilde{\mathbf{U}}(k) \} \]

\[ \mathcal{C} = \{\tilde{\mathbf{x}}(2) = g(\tilde{\mathbf{x}}(1), \tilde{\mathbf{u}}(1), \theta, \mathbf{w}(k)) \}
\]

\[ \tilde{\mathbf{y}}(1) = h(\tilde{\mathbf{x}}(1), \tilde{\mathbf{u}}(1), \theta) + \mathbf{v}(1) \]

\[ \vdots \]

Note that the CSP problem (contracting the domains for the variables involved) becomes larger as the time step advance. Every time step requires an additional vector \( \mathbf{x} \) to be solved, and a number of additional constraints to satisfy. Therefore, the computational complexity of the solution increases with time. An alternative for overcoming this problem is the use of a sliding time window. The time interval from the initial time point to the current one is called time window \( \omega \) [Armengol et al., 2001], Fig. 5.
are actually "branch and prune" algorithms, i.e., algorithms that can be defined as an iteration of two steps [Collavizza et al., 1999]:

1. Pruning the searching space by reducing the intervals associated with the variables until a given consistency property is satisfied.
2. Generating subproblems by splitting the domains of a variable

These techniques can be applied to nonlinear dynamic systems, and their results are not sensitive to strong nonlinearities or nondifferentiabilities in the dynamic system [Jaulin, 2002].

Most interval constraint solvers are based on either hull-consistency (also called 2B-consistency) or box-consistency, or a variation of them [Benhamou et al., 1999]. Box-consistency tackles the problem of hull-consistency for variables with many occurrences in a constraint. The aforementioned techniques are said to be local: each reduction is applied over one domain with respect to one constraint. Better pruning of the variable domains may be achieved if, complementary to a local property, some global properties are also enforced on the overall constraint set.

In this paper, the solution of the fault detection CSP is performed by using the solver RealPaver [Granvilliers and Benhamou, 2006]. BC4 consistency is used for fault detection.

2.3 Summary of main differences

The primary differences between the two approaches are presented in Table 1.

Table 1: Main differences between both techniques.

<table>
<thead>
<tr>
<th>EKF and Z-test</th>
<th>Interval-based Consistency Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty (hypothesis)</td>
<td>Unknown but bounded. Intervals take into account, sensor inaccuracies, measurement noise, unmodeled dynamics, errors in the model, uncertainty in the parameters.</td>
</tr>
<tr>
<td>Model</td>
<td>Not sensitive to strong nonlinearities or nondifferentiabilities in the dynamic system.</td>
</tr>
<tr>
<td>Fault Detection</td>
<td>Mean value of the residuals within a window is out of a threshold (defined by a confidence level). When no solution is found in the CSP.</td>
</tr>
</tbody>
</table>

Regarding the computational complexity, it could be said that the EKF and Z-test are simpler computationally than the interval-based approach. Moreover, the first approach provides a sign corresponding to the direction of change of the observation that may be useful for the fault isolation task [Mosterman and Biswas, 1999].

3 Application Example

An example of a dynamic system based on two interconnected water tanks will be used to compare both techniques.

Figure 6 shows a schematic drawing of the system, which is composed of two tanks, T1 and T2, each having an outflow pipe for draining the tanks P1 and P2, respectively. The first tank also has a source of flow, f_in, for filling the tank.

Figure 6: The two tank system schematic.

3.1 Model equations

The discrete time model obtained from the mass balance considerations is composed of the following discrete time equations:

\[
\dot{x}_1(k+1) = \dot{x}_1(k) + \frac{T_s}{V_1} (f_{in}(k) - k_1 \sqrt{x_1(k)} - k_{12} \text{sgn}(x_1(k) - x_2(k)) \sqrt{x_1(k) - x_2(k)}) + w_1(k),
\]

\[
\dot{x}_2(k+1) = \dot{x}_2(k) + \frac{T_s}{V_2} (-k_2 \sqrt{x_2(k)} + k_{12} \text{sgn}(x_1(k) - x_2(k)) \sqrt{x_1(k) - x_2(k)}) + w_2(k),
\]

\[
\begin{align*}
\hat{f}_1(k) &= k_1 \sqrt{x_1(k)} + v_1(k) \\
\hat{f}_2(k) &= k_2 \sqrt{x_2(k)} + v_2(k)
\end{align*}
\]

The terms \(f_{in}, f_1, f_2, \) and \(f_{12}\) denote the volumetric flows, and \(x_1\) and \(x_2\) are the heights of the water in tanks T1 and T2, respectively. The variables \(f_1, x_1, \) and \(x_2\) are known, and \(f_2\) are known variables obtained from sensors. \(k_1, k_2, k_{12}, \) and \(s_1\) are the constant parameters of the system. \(w_i(k)\) is the perturbation vector at time \(k,\) and it takes into account, for example, an error due to the discretization procedure. \(v_i(k)\) is the measurement noise of the interval measurement \(f_i.\) The sample time, \(T_s,\) is equal to 1 second.

For the EKF scheme, a matrix of partial derivatives (the Jacobian) is computed. At each time step the Jacobian is evaluated with current predicted states. This process essentially linearizes the non-linear model around the current estimate. When the height of the tank T1 is greater than the height of the tank T2, \(x_1(k) > x_2(k),\) the state transition and observation matrices are defined to be the following Jacobians:

\[
F_k = \left. \frac{\partial f}{\partial x} \right|_{x_{k-1}} = \begin{bmatrix}
1 + \frac{T_s}{V_1} \left( -k_1 \sqrt{x_1} - k_{12} \text{sgn}(x_1 - x_2) \right) & \frac{T_s}{V_2} \left( -k_2 \sqrt{x_2} + k_{12} \text{sgn}(x_1 - x_2) \right) \\
\frac{T_s k_1}{2 V_1^{1/2}} & \frac{T_s k_2}{2 V_2^{1/2}}
\end{bmatrix}
\]

\[
H_k = \left. \frac{\partial h}{\partial x} \right|_{x_{k-1}} = \begin{bmatrix}
\frac{T_s}{x_1^{1/2}} & 0 \\
0 & \frac{T_s}{x_2^{1/2}}
\end{bmatrix}
\]
Note that for simplicity in the notation the time step subscript $k$, in variables $x_1$ and $x_2$, are not used in the previous matrices, but they are different at each time step. The fault detection problem of the two tank system can be represented by a CSP. The set of variables is

$$ V = \{ k_1, k_2, k_1, s_2, f_1(k-w), \ldots, f_1(k), f_2(k-w) \ldots f_2(k), \bar{x}_1(k-w), \ldots, \bar{x}_2(k-w), \ldots \bar{x}(k-1), w(k-w), \ldots w(k-1), \bar{w}(k-w), \ldots \bar{w}(k-1) \} $$

the set of domains is

$$ D = \{ K_1, K_2, K_1, S_1, S_2, \bar{F}_1(k-w), \ldots, \bar{F}_1(k), \bar{F}_2(k-w) \ldots \bar{F}_2(k), \bar{X}_1(k-w), \ldots, \bar{X}_2(k-w), \ldots \bar{X}(k-1), \bar{W}(k-w), \ldots \bar{W}(k-1), \bar{V}(k-w), \ldots \bar{V}(k-1) \} $$

where

$$ \bar{X}_1(.) = [0.00, 0.60] $$
$$ \bar{X}_2(.) = [0.00, 0.60] $$
$$ W_1(.) = [-0.001, 0.001] $$
$$ W_2(.) = [-0.001, 0.001] $$

and the set of constraints is

$$ C = \{ f_1(k-w) = k_1 \sqrt{x_1(k-w)} + v_1(k-w) \} $$
$$ f_2(k-w) = k_2 \sqrt{x_2(k-w)} + v_2(k-w) $$
$$ \bar{x}_1(k-w+1) = \bar{x}_1(k-w) + \frac{T_w}{s_1} (f_{wa}(k-w) - k_1 \sqrt{x_1(k-w)} - k_2 \sqrt{x_2(k-w)} + v_1(k-w) + v_2(k-w) $$
$$ \bar{x}_2(k-w+1) = \bar{x}_2(k-w) + \frac{T_w}{s_2} (f_{wa}(k-w) - k_1 \sqrt{x_1(k-w)} - k_2 \sqrt{x_2(k-w)} + v_1(k-w) + v_2(k-w) $$

$$ \bar{x}_1(k) = \bar{x}_1(k-1) + \frac{T_w}{s_1} (f_{wa}(k-1) - k_1 \sqrt{x_1(k-1)} - k_2 \sqrt{x_2(k-1)} + v_1(k-1) + v_2(k-1) $$
$$ \bar{x}_2(k) = \bar{x}_2(k-1) + \frac{T_w}{s_2} (-k_1 \sqrt{x_1(k-1)} + k_2 \sqrt{x_2(k-1)} + v_1(k-1) + v_2(k-1) $$
$$ \bar{f}_1(k) = k_1 \sqrt{x_1(k)} + v_1(k) $$
$$ \bar{f}_2(k) = k_2 \sqrt{x_2(k)} + v_2(k) $$

### 3.2 Simulation results

The scenario considered in this paper deals with a clogging in the output pipe of the tank $T_2$. The fault occurs at time 200s. Two different fault profiles are considered: incipient and abrupt. For the abrupt case, two magnitude deviation are used, a small (5%) change, and the other a larger change (10%) from the nominal value.

White noise (zero mean and standard deviation equal to 1.5% or 3% of the measured signal) was added to the measurements. One hundred runs were conducted for each noise level, fault size, and tuning parameter combination of both (the statistical decision and the interval-based) approaches.

Table 2 lists the parameter values that were used to test the fault detection performance. These parameter values were chosen to show the trade-off of between fault detection and false alarm rate. In the EKF, the process noise covariance matrix $Q$ was varied, and consequently, the gain of the observer, permitting that the actual measurements are “trusted” more and the predicted measurement are trusted less, or the opposite. In the Z-test, the window length $N_2$ was varied between 5 to 9 samples and the confidence level from $2.6\sigma$ to $3\sigma$. Similarly, in the interval-based technique, the measurement intervals and the window length were varied.

<table>
<thead>
<tr>
<th>EKF and Z-test</th>
<th>Interval-based Consistency Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window length</td>
<td>$N_2 = (6, 7, 9)$</td>
</tr>
<tr>
<td>Confidence level</td>
<td>$Z = (2.6, 2.8, 3.0)$</td>
</tr>
<tr>
<td>Window length</td>
<td>$N_1 = 50$</td>
</tr>
<tr>
<td>VarDelay</td>
<td>$= 100$</td>
</tr>
<tr>
<td>Measurement Covariance</td>
<td>$R_{con}$</td>
</tr>
<tr>
<td>Process Noise Covariance</td>
<td>$Q = (0.003, 0.0003, 0.00003)^{2n}$</td>
</tr>
</tbody>
</table>

Table 2: Fault Detection system design parameters.

Fig. 7 and 8 show the fault detection results for both techniques, for an incipient clogging fault in the output pipe $P_2$. In the statistical decision approach, the tuning parameters for this figure are $N_2 = 9$, $z = 3.0$, and $Q = [0.00003]^2$. The vertical lines, in both residuals, indicates the time instant when the fault was detected. In the interval-based approach, the window length is equal to 25, and the interval for the measurement intervals is $\pm 3.0\sigma$.

The mean values of the fault detection times (including the standard deviation in parentheses), the false alarm rates, and the missed alarm rates are reported in Tables 3 and 4.

### Interval-based consistency technique

When the confidence interval of the noise increases, then False Alarm (FA) rate decreases, and Fault Detection (FD) time or Missed Alarm (MA) rate increases. These relationships can be seen in Fig. 9 and 10 (incipient fault), and Fig. 11 (abrupt fault), where there is a trade-off between FD time/MA rate and the FA rate varying the noise interval.

For incipient faults, increasing the window length $w$, decreases the time to FD. In Fig. 12, the results indicate that
As the noise level in the measurements increase, the FD time increases.

**Extended Kalman Filter and Z-test**

Similarly to the previous technique, there is a trade-off between FD time/MA rate and the FA rate, when varying the noise confidence interval of the Z-test, as can be seen in Figs. 13 and 14.

If the window length \( N_2 \) used to compute the mean residual increases, then the FA rate decreases (more smoothing).

Comparing both approaches, the detection times are similar, and regarding the incipient and the abrupt case:

- **Incipient fault.** By looking at the results of both methods with similar FA rates, similar FD times can be achieved but with window length \( \omega \) much greater than the window length \( N_2 \). This is because the EKF stores information from the past, whereas for the interval-based technique, the consistency is checked within a window.

- **Abrupt fault.** EKF and Z-test is more sensitive to smaller magnitudes than the interval based technique, in which the fault is masked by the uncertainty of the model and measurements.
The EKF and Z-test are designed for tracking and fault detection in systems where it is reasonable to capture measurement noise and modeling errors as Gaussian processes. In some situations, when the distributions are not Gaussian [Orlov, 1991] or when the only available information can be expressed as uncertainty bounds (in, e.g., measurements, parameters, and

![Figure 13: Fault detection times using a statistical decision approach. Incipient clogging fault in the output pipe P2.](image)

**4 Conclusions**

The EKF and Z-test are designed for tracking and fault detection in systems where it is reasonable to capture measurement noise and modeling errors as Gaussian processes. In some situations, when the distributions are not Gaussian [Orlov, 1991] or when the only available information can be expressed as uncertainty bounds (in, e.g., measurements, parameters, and

![Figure 14: False alarm rate using a statistical decision approach. Incipient clogging fault in the output pipe P2.](image)
perturbation), interval-based techniques can be used. Computational complexity of EKF and Z-test technique is lower than in the case of interval-based technique, in which bisects have to be performed to obtain less overestimation of the solution.

In general, comparing the performance parameters of the fault detection, both techniques yielded similar fault detection results for a proper combination of tuning parameters. In both techniques, a compromise between false alarms, missed detections and detection delays can be made varying the noise confidence interval.

Finally, in future we intend to also extend this comparison to other fault detection methods as, for example, described in [Niemann et al., 1999; Manders and Biswas, 2003; Baseville and Nikiforov, 1993]. We will perform a formal computational analysis of the false alarm rate to link the parameters of the fault detector for the Z-test with the interval size in the constraint analysis detection method.

5 Acknowledgments

This work has been funded by the Spanish Government (Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica, Ministerio de Educación y Ciencia) through the coordinated research project grant No. DPI2006-15476-C02-02, by the grant No. 2005SGR0296 and the Departament d’Innovació, Universitats, i Empresa of the Government of Catalonia.

Gautam Biswas was partially supported by NASA grants NNX07AD12A and USRA08020-013.

References


Model-based fault detection method using interval analysis:
Application to an aeronautic test bench

Jawad Karim †‡
†CERTIA∗
Department of Automatic Control and R&D
5 rue Des Artisans
93196 Noisy Le Grand, France
jkarim@certia.fr

Carine Jauberthie †and Michel Combacau ‡
‡LAAS-CNRS
Université Paul Sabatier
7, avenue du Colonel Roche
31077 Toulouse Cedex 4, France
{jkarim, cjaubert, combacau}@laas.fr

Abstract
Classical methods for fault detection in industrial automated systems are based on a consistency check between the system behavior and its nominal behavior using the mathematical model. Noise and disturbances acting on the system are often modeled by probability distributions. However, it is not always possible to acquire information on noise and disturbances. The modeling in a bounded-error context can be a solution.

This paper concerns the fault detection in an aeronautic test bench in a bounded-error context. The serviceability of the test bench is estimated from a comparison between the measured system outputs and predicted outputs by model, where the uncertainties of modeling and the measurement noise are supposed bounded.

1 Introduction
One of the major challenges of monitoring and supervision systems consists in increasing the safety and the availability of the industrial installation. Early detection of any behavior deviation of a machine compared to the nominal one is the necessary first step for the implementation of preventive and corrective actions. Indeed, these processes introduce a high level of difficulty for control/monitoring because of their dynamical character, of the high number of variables to be dealt with and of the complexity of their analytical representation [Karim et al., 2007]. This complexity is accentuated by signal noise, disturbances, incomplete knowledge and parameters uncertainties. Complex systems are often subject to uncertainties that make awkward the modeling step. These uncertainties can be unstructured when the equations of the system are not entirely known or structured when the equations are known but not the values of their parameters. In both cases, it is particularly difficult to get an accurate model of the perturbations and noise acting on the system. This may turn the usual stochastic framework inappropriate [Benazera et al., 2002], [Williams and Nayak, 1999].

Model-based detection in a bounded-error context is an interesting alternative to stochastic model based detection when uncertainties, perturbations and noise are assumed to be bounded but otherwise unknown. Tools set computation were developed in frame of interval arithmetic [Moore, 1966]. In automatic control community, they are used to solve problems of state and/or parameters estimation in a bounded-error context. These methods are now subject of a growing interest in various communities and are applied to many tasks, for example [Berz and Makino, 1998], [Corliss, 1994], [Raisi et al., 2004], [Sigla, 1999], [Ramdani, 1995] or [Ribot et al., 2007]. However, they still suffer from two main drawbacks: uncertainty propagation is difficult to manage and the resulting algorithms are often of high complexity.

In this paper the mean of detection is performed in a bounded-errors context. We consider model-based fault detection of the process in which the parameters are described by a set of bounded variables. This class of systems have outputs described by an acceptable domain including all possible normal behaviors. Interval arithmetic is used to describe the whole continuous range of behaviors represented by an interval model. The main idea of the proposed approach is to apply the model to provide the bounded behavior of a given interval parameter and to use it in the fault-detection system as a residual generator. In this way, the model outputs are insensitive to the influence of disturbances and uncertainties. With this modeling approach we can represent the system behavior with a certain degree of abstraction. Thus we will have more robustness and better adequacy with the requirement of supervision.

The article is organized as follows. The next section provides an overview of interval analysis, its original purpose and its use for fault detection. In the third section, the case

*Center of Studies and Research in Industrial and Applied Techniques.
study is presented: we describe the aeronautic test bench and we explain why the faults must be detected in real-time. The application and the obtained results are analyzed. Finally some conclusions are outlined in the last section.

2 Fault detection using intervals

The interval analysis has been introduced by R. Moore [Moore, 1966]. The main idea is to reason about intervals instead of real numbers. The intervals can be used with consideration to uncertainties on measured values or on model parameters. The aim is firstly to guarantee results by calculating a set which contains the effective solution and secondly to provide the adequate bounds of the solution. These results are then guaranteed. The width of the set and the result precision, may be chosen according to various criteria among which response time or computation cost [Neumaier, 1990], [Rihm, 1994].

In the following section, we present some basics tools of interval analysis.

2.1 Basic tools of intervals analysis

Basic definitions – A real interval \([x, x]\) is a closed, bounded and connected subset of \(\mathbb{R}\) and it is denoted:

\[
x = [x, x] = \{ x \in \mathbb{R} \mid x \leq x \leq x \},
\]

where \(x\) represents the lower bound of \([x, x]\) and \(x\) represents the upper bound. The set of all real intervals of \(\mathbb{R}\) will be denoted by \(\mathbb{I}\).

The width of an interval \([x, x]\) is defined by \(w([x, x]) = x - x\), and its midpoint by \(m([x, x]) = \frac{x + x}{2}\). Two intervals \([x, x]\) and \([y, y]\) are equal if and only if \(x = y\) and \(x = y\).

Interval Vector – An interval vector (or box) \([X]\) is a vector with interval components and may equivalently be seen as a cartesian product of scalar intervals:

\[
[X] = [x_1] \times [x_2] \times \cdots \times [x_n].
\]

The set of \(n\)-dimensional real interval vectors is denoted by \(\mathbb{I}^n\).

An interval matrix is a matrix with interval components. The set of \(n \times m\) real interval matrices is denoted by \(\mathbb{I}^{n \times m}\). The width \(w(.)\) of an interval vector (or of an interval matrix) is the maximum of the widths of its interval components:

\[
w([X]) = \max_i (x_i - x_i), \quad i = 1, \cdots, n.
\]

The midpoint \(m(.)\) of an interval vector (resp. an interval matrix) is a vector (resp. a matrix) composed of the midpoint of its interval components:

\[
m([X]) = \left( \frac{(x_1 + x_1)}{2}, \ldots, \frac{(x_n + x_n)}{2} \right).
\]

Interval arithmetic – The arithmetical operations \(\{+,-,*,/\}\) can be extended to intervals [Jaulin et al., 2001].

Arithmetic operations on two intervals \([x, x]\) and \([y, y]\) can be defined by:

\[
of \in \{+,-,*,/\}, \quad [x, y] = \{ x \circ y \mid x \in [x, y], y \in [y, y] \}.
\]

It is easy to prove that the set of real compact intervals is closed with respect to these operations.

The following rules hold:

\[
[x, y] + [y, y] = [x + y, x + y],
\]

\[
[x, y] - [y, y] = [x - y, x - y],
\]

\[
[x, y] \ast [y, y] = [\min(x, x, y, y), \max(x, x, y, y)],
\]

\[
n(x) = \begin{cases} x + \frac{1}{\|y\|}, & \text{if } 0 \notin [y], \\ [-\infty, \infty], & \text{else.} \end{cases}
\]

All these operations are inclusion monotonic [Moore, 1979]. It means that:

\[
\text{if } [x] \subset [x] \text{ and } [y] \subset [y], \text{ then } [x] \circ [y] \subset [x] \circ [y].
\]

Extension to interval vectors and interval matrices is trivial.

Inclusion function – Classical operations for interval vectors (resp. interval matrices) are direct extensions of the same operations for punctual vectors (resp. punctual matrices).

Let \(f : \mathbb{I}^n \to \mathbb{I}^m\), the range of the function \(f\) over an interval vector \([u]\) is given by:

\[
f([u]) = \{ f(x) \mid x \in [u] \}.
\]

The interval function \(f\) from \(\mathbb{I}^n\) to \(\mathbb{I}^m\) is an inclusion function for \(f\) if:

\[
\forall [u] \in \mathbb{I}^n, \quad f([u]) \subseteq [f([u])].
\]

An inclusion function of \(f\) can be obtained by replacing each occurrence of a real variable by its corresponding interval and by replacing each standard function by its interval evaluation. Such a function is called the natural inclusion function. In practice the inclusion function is not unique, it depends on the syntax of \(f\).

2.2 Fault detection in a bounded-error context

Fault detection on industrial processes is usually solved in a stochastic framework in which the measurement noise and disturbances are modeled by a priori known probability laws. On the contrary, in the proposed approach, measurement and modeling errors are supposed unknown, but varying between bounded that are known or, at least, a priori acceptable.

In this work the normal behavior is obtained by a real-time simulation of the interval model that consider the uncertainties in the model parameters and sensor measurements. Some model parameters are supposed varying between upper and lower bounds [Bernard and Gouzé, 2004], [Rapaport and Dochain, 2005], [Gouzé et al., 2000]. Thus, the predicted output \(y_m\) also belongs to an interval noted \([Y_m]\).

The detection occurs naturally on the comparison between the interval predicted behavior obtained from a representation
model (normal functioning model) with the presence of uncertainties and the behavior of the real system in which faults can occur.

The experimental measurements $Y_{exp}$ are related to the corresponding predicted behavior from the interval model $[Y_m]$ by the following relation:

- Case of normal functioning: $Y_{exp} \in [Y_m]$, 
- Case of abnormal functioning: $Y_{exp} \notin [Y_m]$.

the interval model as represent the system adequately while taking into consideration the uncertainties and noise, they would signified the non-existence of the solution outside the bounds of interval solution.

3 Case study

Our case study is an aeronautic test bench whose goal is to simulate the aerodynamic efforts and mechanical frictions during the deployment operation of an aircraft actuator. In doing so, we want to check their aptitude to withstand the important constraints undergone during the flight.

3.1 System description

The bench consists of several distinct parts. A schematic drawing of the system is presented in Figure 2. The bench is constituted of the following parts:

- A mechanical frame: Its structure will be modeled, simulated and calculated to resist to the maximum efforts produced by the actuators (finite element method);
- A Trolley has been realized. Its translations are guided by rails and ball-bearings. This system is pulled by the aircraft actuator;
- A load system composed of one servo-jack (hydraulic actuators) under effort control, reproducing the aerodynamic efforts applied to the aircraft actuator system;
- The sensors: Effort, velocity, pressure, displacement, flow (the number of sensors could be increased);
- Power components (electric and hydraulic generation) needed to provide the energy required by the test rig;
- A data-processing and controller part for control, measurements and acquisition.

The aircraft actuator system ensuring the displacement of the moving body (trolley), the main goal of the bench is to recreate the aerodynamic effort. The specific load profiles to each operational mode must be tracked by the actuator load.

3.2 Physical modeling

The development of an accurate dynamic model for an aeronautics test bench is important for understanding the system and for developing a robust model-based fault detection. This step is strongly based on the modeling software Matlab®/Simulink®, which allow a systematic generation of standalone application on a real time computer by using RTW®/xPCTarget® (Real-Time kernel).

As a first step, the dynamic model is used as a support for analysis, test and validation of the detection approach in a virtual environment. It will be used in the second step to finalize the implementation on the test bench. To this end, a description of the dynamics for the servovalve, the hydraulic cylinder and the mechanical frame (stiffness and mass of mechanical frame) is required.

Servovalve

A servovalves provide closed loop flow or pressure response to an electronic control signal. It is used to control the position, velocity, or force of a hydraulic actuator load. This is including a sliding spool that receives the linear motion of the torque motor, which is modeled as a second order system with a specified frequency and damping. The spool displacement $X_s$ modulates the orifice area, which in turn affects the output dynamic flow, and is controlled by an input current command.

The relationship between dynamic flow characteristics of an electro-hydraulic servovalve, spool displacement and pressure supply (variable pressure pump), have been determined from the nonlinear equations.

For $X_s \geq 0$ :

$$Q_1 = \sqrt{\frac{2}{\rho_s}} \frac{K}{X_s} \left| P_{al} - P_1 \right| \text{sign}(P_{al} - P_1),$$
$$Q_2 = \sqrt{\frac{2}{\rho_s}} \frac{K}{X_s} \left| P_2 - P_1 \right| \text{sign}(P_2 - P_1).$$

(1)

For $X_s < 0$ :

$$Q_1 = \sqrt{\frac{2}{\rho_s}} \frac{K}{X_s} \left| P_1 - P_{al} \right| \text{sign}(P_1 - P_{al}),$$
$$Q_2 = \sqrt{\frac{2}{\rho_s}} \frac{K}{X_s} \left| P_2 - P_{al} \right| \text{sign}(P_2 - P_{al}).$$

(2)
In Eqs. (1-2), \( X_s \) is the spool displacements, \( Q_1 \) (respectively \( Q_2 \)) is the flow control output ports 1 (respectively 2), \( K \) is the flow coefficient depending on the orifice area, \( \xi \) is the hydraulic damping factor, \( \rho \) is the density. The variables \( P_{at}, P_{o} \) are respectively supply pressure and outlet pressure; \( P_1 \) and \( P_2 \) are the pressure in cylinder chambers.

**Hydraulic cylinder**

The hydraulic cylinder is \( l_l \) double-acting, double rod. This kind of cylinder gets its power from pressurized hydraulic fluid and generates the same effort following a two-way linear displacement. It consists of a rod: a circular piston could slide into a cylindrical body. The piston separates the body into two cylinder chambers; an inflow in a chamber entails moving piston and rod, and creates a return flow from the second chamber.

The hydraulic actuator model is characterized by three dynamic states (the load actuator displacement \( x_{la} \), the velocity \( v_{la} \) and the force between rod and aircraft actuator), two inputs (flow control from servovalve), and one output (the hydraulic force \( F_{hyd} \) acting on the mechanical frame).

**Dynamic equation:**

\[
M\ddot{x}_{la} = (P_1 - P_2)S - (2f_{s1} + 2f_{s2})\text{sign}(\dot{x}_{la}) - (2f_{f1} + 2f_{f2})\dot{x}_{la}. 
\] (3)

**Flow control equations:**

\[
Q_1 = S\dot{x}_{la} + \frac{V_{01}}{B} \frac{dP_1}{dt} + Q_{int}. 
\] (4)

\[
Q_2 = S\dot{x}_{la} - \frac{V_{02}}{B} \frac{dP_2}{dt} + Q_{int}. 
\] (5)

**Hydraulic force:**

\[
F_{hyd} = (P_1 - P_2)S = \Delta P \cdot S . 
\] (6)

In Eqs (3-6), \( f_{s1} \) and \( f_{s2} \) (\( f_{f1} \) and \( f_{f2} \)) are respectively the dry friction (viscous friction) between piston/body and rod/piston. \( Q_1 \) and \( Q_2 \) are the inflow/outflow in the cylinder chambers, \( P_1, P_2 \) are the pressure cylinder chambers, \( B \) is the fluid bulk modulus, \( V_{01}, V_{02} \) are respectively the load actuator volume in chamber 1 and 2. \( Q_{int} \) is the leakage flow between the two chambers, \( S \) is the load actuator area and \( M \) is the moving mass of the load actuator.

**Mechanical frame**

We have put in a simplified form the stiffness of aircraft real inertia. We have also taken into account the distributed mass on the mechanical elements, stiffness between aircraft actuators and load actuator.

The expression of dynamic stiffness is given by the following equations:

- **Side actuator load:**

\[
\frac{dF_{la}}{dt} = k_{la}(V_{la} - V_m). 
\] (7)

- **Side aircraft actuator:**

\[
\frac{dF_{sc}}{dt} = k_{sc}(V_m - V_{sc}), 
\] (8)

where \( k_{la} \) and \( k_{sc} \) are respectively the load actuator stiffness side and actuator aircraft stiffness side. \( F_{la} \) and \( F_{sc} \) are the force acting on the mechanical frame by actuator load and aircraft actuator, \( V_{la}, V_m, W_m \) and \( V_{sc} \) are the velocity in the several parts.

The expression of dynamic inertia:

\[
V_m = \frac{W_m}{r_{la}} = \frac{W_m}{r_{sc}}. 
\] (9)

\[
\int \frac{dW_m}{dt} = \frac{F_{la}}{r_{la}} = \frac{F_{sc}}{r_{sc}}. 
\] (10)

with \( r_{la} \) and \( r_{sc} \) representing the lever arm attached to mechanical frame inertia.

### 3.3 Results analysis

In general, either normal or abnormal behavior of process can be modeled using elementary analytical relations derived from physics. To take the uncertainties of the system into account, we have represented all the variables and the parameters by intervals. The bounded values are based on the specifications given by the suppliers.

The fault detection of the proposed model-based fault detection method is evaluated on the full emulation process. The emulator is a model of the bench on which we can easily simulate the occurrence of faults.

The resulting output trajectories are given in figures 3 and 4. In these figures, the plain lines represent the measured force and the dotted lines represent the reconstruction of this force by using the interval model. The test duration is set to 3 seconds.

In figure 3, the system is simulated without any failure. Indeed, the obtained envelopes for state variables show that uncertainty is appropriately controlled by the interval model. The fault the most difficult to be detected on a hydraulic cylin-

![Figure 3](image_url)

**Figure 3:** The force reconstruction defined by the error-bounded envelopes.
The consistency between the interval model and the measured value is checked by the error-bounded estimates of the band, and therefore, a fault is detected when the measurement does not between the external estimates. The fault is detected at 2.55 s.

The tests show good results for detecting a fault. This algorithm is very efficient because it is based on checking the deviations between the actual value of a measure and the predicted envelopes obtained by interval tool. Moreover, the use of the model interval analysis guarantees that this method reduces the number of false alarms and the occurrence of gradual faults.

4 Conclusion

In this work, fault-detection results on an aeronautic test-bench are presented. The proposed approach for fault detection is based on interval analysis which provides guaranteed results in an error bounded context.

The advantage of the interval approach is to obtain the consistency between a model and a system. The first results are promising. Future works will be devoted to fault diagnosis. Indeed, when the fault is detected, the measurement is either larger or smaller than the estimated bound. This information, combined with qualitative information from the model, can be used for diagnosis.

References


Model-based Fault Detection Method Using Interval Analysis: Application to an Aeronautic Test Bench

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
Automated Learning of Diagnosis Models for Component-Oriented Robot Control Software

Alexander Kleiner 1 and Gerald Steinbauer 2 and Franz Wotawa 2 *

1 Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, D-79110 Freiburg, Germany
kleiner@informatik.uni-freiburg.de

2 Institute for Software Technology, Graz University of Technology
Infeldgasse 16b/II-8010, Austria
{steinbauer,wotawa}@ist.tugraz.at

Abstract

Control software of autonomous mobile robots comprises a number of software modules which show very rich behaviors and interact in a very complex manner. These facts among others have a strong influence on the robustness of robot control software in the field. In this paper we present an approach which is able to automatically derive a model of the structure and the behavior of the communication within component-orientated control software. Such a model can be used for on-line model-based diagnosis in order to increase the robustness of the software by allowing the robot to autonomously cope with faults occurred during runtime. Due to the fact that the model is learned from recorded data and the use of the popular publisher-subscriber paradigm the approach can be applied to a wide range of complex and even partially unknown systems.

1 Introduction

Control software of autonomous mobile robots comprises a number of software modules which show very rich behaviors and interact in a very complex manner. Because of this complexity and other reasons like bad design and implementation there is always the possibility that a fault occurs at runtime in the field. Such faults can have different characteristics like crashes of modules, deadlocks or wrong data leading to a hazardous decision of the robot. This situation can occur even if the software is carefully designed, implemented and tested. In order to have truly autonomous robots operating for a long time without or with limited possibility for human intervention, e.g., planetary rovers exploring Mars, such robots have to have the capability to detect, localize and to cope with such faults.

In [Steinbauer and Wotawa, 2005; Steinbauer et al., 2006] the authors presented a model-based diagnosis framework for control software for autonomous mobile robots. The control software is based on the robot control framework Miro [Utz et al., 2002; Utz, 2005] and has a client-server architecture where the software modules communicate by exchanging events. The idea is to use the different communication behaviors between the modules of the control software in order to monitor the status of the system and to detect and localize faults. The model comprises a graph specifying which modules communicate with each other. Moreover, the model has information about the type of a particular communication path, e.g., whether the communication occurs on a regular basis or sporadically. Finally, the model includes information about which inputs and outputs of the software modules have a functional relation, e.g., which output is triggered by which input. The model is specified by a set of logic clauses and uses a component-based modeling schema [Friedrich et al., 1999]. Please refer to [Steinbauer and Wotawa, 2005; Steinbauer et al., 2006] for more details.

The diagnosis process itself uses the well known consistency-based diagnosis techniques of Reiter [Reiter, 1987]. The models of the control software and the communication were created by hand by analyzing the structure of the software and its communication behavior during runtime. Because of the complexity of such control software or the possible lack of information about the system it is not feasible to do this by hand for large or partially unknown systems.

Therefore, it is desirable that such models can be created automatically either from a formal specification of the system or from observation of the system. In this paper we present an approach which allows the automatic extraction of all necessary information from the recorded communication between the software modules. The algorithm provides all information needed for model-based diagnosis. It provides a communication graph showing which modules communicate, the desired behavior of the particular communication paths and the relation between the inputs and outputs of the software modules.

These model learning approach was originally developed and tested with the control software of the Lurker robots [Kleiner and Dornhege, 2007] used in the RoboCup rescue league. This control software uses the IPC communication framework [Simmons, 1994], which is a very popular event-based communication library used by a number of robotic research labs worldwide. However, the algorithm simply can be adapted to other
event-based communication frameworks, such as for instance Miro. The next section describes in more detail how the model is extracted from the observed communication.

2 Model Learning

Control systems based on IPC use an event-based communication paradigm. Software modules which want to provide data are publishing an event containing the data. Other software modules which like to use this data subscribe for the appropriate event and get automatically informed when such an event is available. A central software module of IPC is in charge for all aspects of this communication. Moreover, this software module is able to record all the communication details. It is able to record the type of the event, the time the event was published or consumed, the content of the event, and the names of the publishing and the receiving module. The collected data is the basis for our model learning algorithm. Formally an event is defined as following:

Definition 1 (Event) An event \( e \) is a tuple \((l_p, t_p, d, P, C)\) where:

- \( l \) is the label or name of the event \( e \)
- \( t_p \) is the time the event was published
- \( t_C \) is the time the event was consumed
- \( d \) is the data or payload of the event
- \( P \) is the publisher of the event
- \( C \) is the consumer of the event

Please note that if an event is consumed by multiple consumers for each consumption such an event is recorded.

Figure 1 depicts such collected data for a small example control software comprising only 5 modules with a simple communication structure. This example will be used in the following description of the model learning algorithm. The control software comprises two data path. The one is the path for the self-localization of the robot. The two modules in the path Odometry and Self-Loc provide data on a regular basis. The other is the path for object tracking. The module Vision provides new data on a regular basis. The module Tracker provides data only if new data is available form the module Vision. The figure shows when the different events were published. Based on this recorded communication we extract the communication model step by step.

2.1 The communication graph

At a first step the algorithm extract a communication graph from the data. The nodes of the graph are the different software modules. The edges represent the different events which are exchanged between the modules. Each event is represented by at least one edge. If the same event is received by multiple modules, there is an edge to every receiving module originating from the publishing module. Figure 2 depicts the communication graph for the above example. This graph shows the communication structure of the control software. Moreover, it shows the relation of inputs and outputs of the different software modules because each node knows its connections. Such a communication graph is not only useful for diagnosis purposes, but it is also able to expressively visualize the relation of modules of larger or partially unknown control software.

Formally the communication graph can be defined as following:

Definition 2 (CG) A communication graph (CG) is a directed graph with the set of nodes \( M \) and the set of labeled edges \( C \) where:

- \( M \) is a set of software modules sending or receiving at least one event.
- \( C \) is a set of connections between modules, the direction of the edge points from the sending to the receiving module, the edge is labeled with the name of the related event.

Please note that the communication graph may contain cycles. Usually such cycles emerge from acknowledgement mechanisms between two modules.

The algorithm for the creation of the communication graph can be formalized as following:

computeGraph

Input: a set of recorded events \( E \)
Output: a set of nodes \( M \) and edges \( C \)

1. Let \( M \) be the empty set.
2. Let \( C \) be the empty set.
3. For all \( e \in E \):
   (a) If \( p(e) \notin M \) add \( p(e) \) to \( M \).
   (b) If \( c(e) \notin M \) add \( c(e) \) to \( M \).
   (c) If \((p(e), c(e), l(e)) \notin C\) add \((p(e), c(e), l(e))\) to \( C \).
4. Return \( M \) and \( C \).

The algorithm starts with an empty set of nodes \( M \) and edges \( C \). The algorithm iterations through the set \( E \) of all recorded communication events. If either the sender or the receiver is not in the set of the nodes the sender or the receiver is added to the set. If there is no edge pointing form the sending to the receiving node with the proper label a new edge with the appropriate label is added between the two modules. The functions \( p(e), c(e), l(e) \) return the publisher, the consumer and the label of an event \( e \).

Moreover, we define the two functions \( in : CO \rightarrow 2^C \) and \( out : CO \rightarrow 2^C \) which return the edges pointing to and from a node.

2.2 The communication behavior

In a next step the behavior or type of each event connection is determined. For this determination we use the information of the node the event connection comes from, and the recorded information of the event related to the connection, and all events related to the sending node.
We can distinguish the following types: triggered event connection (1), periodic event connection (2), bursted event connection (3) and random event connection (4). In order to describe the behavior of a connection formally we define a set of connection types

\[ CT = \{ \text{periodic}, \text{triggered}, \text{bursted}, \text{random} \} \]

and a function

\[ ctype : C \mapsto CT \]

which returns the type of a particular connection \( c \in C \).

The type of an event connection is determined by tests like estimations on the mean and the standard deviation of the time between the occurrence of the events on the connection, and comparison or correlation of the occurrence of two events. The criteria used to assign an event connection to one of the four categories are summarized below:

**triggered** A triggered event only occurs if its publishing module recently received a trigger event. In order to determine if an event connection is a triggered event connection, the events on connection \( c \in out(m) \) are correlated to the events on the set of input connection to the software module \( f = in(m) \). If the number of events on connection \( c \), which are correlated with an event on a particular connection \( t \in in(m) \), exceed a certain threshold, connection \( t \) is named as trigger of connection \( c \). The correlation test looks for the occurrence of the trigger event prior the observed event. Note each trigger event can only trigger one event. If connection \( c \) is correlated with at least one connection \( t \in in(m) \) connection \( c \) is categorized as a triggered connection. Usually, such connections are found in modules doing calculations only if new data are available.

**periodic** On a periodic event connection the same event regularly occurs with a fixed frequency. We calculate from the time stamps of the occurrence of all events a discrete distribution of the time difference between two successive events. If there is high evidence in the distribution for one particular time difference, the connection is periodic with a periodic time of the estimated time difference. For a pure periodic event connection one gets a distribution close to a Dirac impulse. Usually, such connections are found with modules providing data at a fixed frame rate, such as a module sending data from a video camera.

**bursted** A bursted event is similar to the periodic event but its regular occurrence can be switched on and off for a period of time. A event connection is classified as bursted if there exist time periods where the criteria of the periodic event connection hold. Usually, such connections are found with modules which do specific measurements only if the central controller explicitly enables them, e.g., a complete 3d laser scan.

**random** For random event connections none of the above categories match and therefore no useful information about the behavior of that connection can be derived. Usually, such connections are found in modules which provide data only if some specific circumstance occur in the system or its environment.

In the case of the above example, the algorithm correctly classified the event connections odometry, objects and pose as periodic and the connection velocity as triggered with the trigger objects.

It has to be noted that the quality of the classification of the communication behavior depends on parameters like thresholds. Currently, these parameters are tuned...
by hand. In the future a more systematic evaluation of
the influence of the parameters has to be done.

2.3 The observers
The algorithm instantiates an observer for each event connection in order to be able to monitor the actual behavior of the control software. The type of the observer is determined by the type of the connection and its parameters, estimated by the methods described before. An observer raises an alarm if there is a significant discrepancy between the currently observed behavior of an event connection and the behavior learned beforehand during normal operation. The observer provides as an observation \( O \) the atom \( ok(l) \) if the behavior is within some tolerance and the atom \( \neg ok(l) \) otherwise. Where \( l \) is the label of the corresponding edge in the communication graph. The observations of the complete control software \( OBS \) are the union of all individual observations

\[
OBS = \bigcup_{i=1}^{n} O_i
\]

Where \( n \) is the number of observers.

The following observers are used:

**triggered** This observer raises an alarm if within a certain timeout after the occurrence of a trigger event no corresponding event occurs or if the trigger event is missing prior the occurrence of the corresponding event. In order to be robust against noise, the observer uses a majority vote for a number of succeeding events, e.g., 3 votes.

**periodic** This observer raises an alarm if there is a significant change in the frequency of the events on the observed connection. The observer checks if the frequency of successive events does vary too much from the specified frequency. For this purpose, the observer estimates the frequency of the events within a sliding time window.

**burst** This observer is similar to the observer above. It differs in the fact that this observer starts the frequency check only if events occur and does not raise an alarm if no events occur.

**random** This is a dummy observer which always provides the observation \( ok(l) \). This observer is implemented for completeness.

2.4 The system description
The communication graph together with the type of the connections is a sufficient specification of the communication behavior of the robot control software. This specification can be used in order to derive a system description for the diagnosis process. It is a description of the desired or nominal behavior of the system. In order to be able to be used in the diagnosis process, the system description is automatically written down as a set of logical clauses. This set can easily be derived from the communication graph and the classification of the behavior of the connections.

The algorithm to derive the system description can be formalized as following:

**computeSD**

\( Input \): the communication graph with nodes \( M \) and connections \( C \)

\( Output \): a set of clauses

1. Let \( SD \) be the empty set.
2. For all \( c \in C \):
   If \( host(p(c)) \neq host(c(c)) \)
   (a) If \( c_{type}(c) = triggered \) add
   \[
   \neg AB(p(c)) \land \bigwedge_{t \in trigger(c) \land t \in in(p(c))} ok(t) \land \\
   \neg AB(host(p(c))) \land AB(host(c(c))) \rightarrow ok(c)
   \]
   to \( SD \)
   Else add
   \[
   \neg AB(p(c)) \land \neg AB(host(p(c))) \land \neg AB(host(c(c)))
   \]
   \( \rightarrow ok(c) \)
   to \( SD \)
   Else
   (b) If \( c_{type}(c) = triggered \) add
   \[
   \neg AB(p(c)) \land \bigwedge_{t \in trigger(c) \land t \in in(p(c))} ok(t) \rightarrow ok(c)
   \]
   to \( SD \)
   Else add
   \[
   \neg AB(p(c)) \rightarrow ok(c)
   \]
   to \( SD \)
3. For all \( m \in M \):
   Add
   \[
   \bigwedge_{c' \in out(m)} ok(c') \rightarrow \neg AB(m)
   \]
   to \( SD \)
4. Return \( SD \).

The functions \( p(c) \) and \( c(c) \) returns the publishing and receiving module of an event connection \( c \). The function \( host(m) \) returns the host a particular module \( m \) is running on. The algorithm starts with an empty set \( SD \). For every event connection in two steps, clauses are added to the system description. In the first step, a clause for forward reasoning is added. The clause specifies if a module works correct and all related inputs and outputs behave as expected. Depending on the type of the connection, we add the following clause to \( SD \). If connection \( c \) is \( triggered \), we add a clause that states that if the module and all related inputs work as expected, also the output has to work as expected. Otherwise a clause is added that states if the module works as expected also the output has to work as expected (see Line 2). \( \neg AB(m) \) means that the module \( m \) is not abnormal and the module works as expected. The atom \( ok(c) \) specifies that the connection \( c \) behaves as expected. Moreover, if the hosts of the sending and receiving module of

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
a connection $c$ are different a fact that the network interfaces of these modules have to work correct is added, e.g., $\neg AB(\text{host}(p(c)))$.

In a second step, a clause for backward reasoning is added. The clause specifies if all output connections $c'$ of module $m$ behave as expected, the module itself has to behave as expected (see Line 3).

Figure 3 depicts the system description obtained for the above example control software.

3 Model-based diagnosis

For the detection and localization of faults we use the consistency-based diagnosis technique of [Reiter, 1987]. A fault detectable by the derived model causes a change in the behavior of the system. If such an inconsistency between the modeled and the observed behavior emerges a failure has been detected. Formally, we define this by:

$$SD \cup OBS \cup \{\neg AB(m) | m \in M \} \models \bot$$

The latter set expresses the fact that we assume that all modules work as expected.

In order to localize the module responsible for the detected fault we have to calculate a diagnosis $\Delta$. Where $\Delta$ is a set of modules $m \in M$ we have to declare as faulty (change $\neg AB(m)$ to $AB(m)$) in order to resolve the above contradiction. We use our implementation\(^1\) of this diagnosis process for the experimental evaluation of the models. Please refer to [Steinbauer and Wotawa, 2005; Steinbauer et al., 2006] for the detail of the diagnosis process.

4 Experimental Results

In order to show the potential of our model learning approach, the approach has been tested on three different types of robot control software. We evaluated whether the approach is able to derive an appropriate model reflecting all aspects of the behavior of the system. The derived model was evaluated by the system engineer who has developed the system. Moreover, we injected artificial faults like module crashes in the system, and evaluated if the fault can be detected and localized by the derived model.

4.1 A small example control software

The example software from the introduction comprises five modules. The module Odometry provides odometry data at a regular basis. This data is consumed by the module SelfLoc, which does pose tracking by integrating odometry data, and providing continuously a pose estimate to a visualization module User. The module Vision provides position measurements of objects. The module Tracker uses these measurements to estimate the velocity of the objects. New velocity estimations are only generated if new data is available. The velocity estimates are also visualized by the GUI. Figure 1 shows the recorded communication of this example. Figure 2 depicts the communication graph extracted from the recorded data.

It correctly represents the actual communication structure of the example, and shows the correct relation of event producers and event consumers.

Moreover, the algorithm correctly identified the type of the event connections. This can be seen by the system description the algorithm has derived which is depicted in Figure 3. It also instantiates the correct observer for the four event connections. A periodic event observer was instantiated for odometry, objects and pose, and a triggered event observer was instantiated for velocities.

1. $\neg AB($ Vision $)$ $\rightarrow$ ok(objects)
2. $\neg AB($ Odometry $)$ $\rightarrow$ ok(odometry)
3. $\neg AB($ Tracker $) \land$ ok(objects) $\rightarrow$ ok(velocities)
4. $\neg AB($ SelfLoc $)$ $\rightarrow$ ok(pose)
5. ok(objects) $\rightarrow$ $\neg AB($ Vision $)$
6. ok(odometry) $\rightarrow$ $\neg AB($ Odometry $)$
7. ok(velocities) $\rightarrow$ $\neg AB($ Tracker $)$
8. ok(pose) $\rightarrow$ $\neg AB($ SelfLoc $)$

Figure 3: The system description automatically derived for the example control software.

Figure 3 depicts the extracted system description. Clauses 1 to 4 describe the forward reasoning. Clauses 5 to 8 describe the backward reasoning. Clause 3 states that the module Tracker works correctly if a velocity event occurs only after a trigger event. For instance, Clause 6 states that if all output connections of module Odometry work as expected, consequently the module itself works correct. This automatically generated system description was used in some diagnosis tests. We randomly shutdown modules and evaluate if the fault was correctly detected and localized. For this simple example the faults were always proper identified.

4.2 Autonomous exploration robot Lurker

In a second experiment we recorded the communication of the control software of the rescue robot Lurker [Kleiner and Dornhege, 2007] while the robot was autonomously exploring an unknown area. The robot is shown in Figure 4.

Figure 4 depicts the autonomous rescue robot Lurker of the University of Freiburg.

The control software of this robot is far more complex.

---

\(^1\)The implementation can freely be downloaded at http://www.ist.tugraz.at/mordams/.

as in the example above since it comprises all software modules enabling a rescue robot to autonomously explore an area after a disaster. Figure 5 shows the communication graph derived from the recorded data, clearly showing the complex structure of the control software. The numbers in the labels of the edges denote the average frequency of events on the connections. Please note that a frequency of 0 Hz means the actual frequency is below 1 Hz. That is just a reduction of the resolution for presentation purpose.

From the communication graph and the categorized event connections a system description with 70 clauses with 51 atoms and 35 observers was derived. The extraction of the model from recorded data with a size of 87 MByte took 28 s on a computer equipped with a dual-core CPU running at 1.8 GHz and 1 GByte of memory. After a double check with the system engineer of the control software it was confirmed that the automatically derived model maps the behavior of the system.

4.3 Teleoperation Telemax robot.

In a final experiment we record data during a teleoperated run with the bomb-disposal robot Telemax. The robot Telemax is shown in Figure 6.

![Figure 6: The teleoperated robot Telemax.](image)

Figure 7 depicts the communication graph derived from the recorded data. It clearly shows that the control software for teleoperation shows a far less complex communication structure than in the autonomous service. From the communication graph and the categorized event connections a system description with 44 clauses with 31 atoms and 22 observers was derived.

5 Related Research

There are many proposed and implemented systems for fault detection and localization in autonomous systems. The Livingstone architecture by Williams and colleagues [Muscettola et al., 1998] was used on the space probe Deep Space One to detect failures in the probe’s hardware and to recover from them.

Model-based diagnosis also has been successfully applied for fault detection and localization in digital circuits and car electronics and for software debugging of VHDL programs cited in [99].

In [Micalizio et al., 2006] the authors show how model-based reasoning can be used for diagnosis for a group of robots in the health care domain. The system model comprises interconnected finite state automata.

In [Grosclaude, 2004] a model-based approach for monitoring of component-based software was presented. The behavior of software components were modeled by Petri nets. Places in the net represent the state of a component. Transitions model the interactions with other components. These interactions, sending and receiving of messages, were used to locate misbehaviors in a software component.

Verma and colleagues [Verma et al., 2004] used particle filter techniques 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 advantage of this approach is that it accounts uncertainties of the robot’s sensing and acting and in its environment because the most probable state is derived from unreliable measurements.

6 Conclusion and Future Work

In this paper we presented an approach which allows the automated learning of communication models for robot control software. The approach uses recorded event communication. The approach is able to automatically extract a model of the behavior of the communication within component-orientated control software. A clear benefit of the approach is the fact that it is able to derive a communication model even if the information about the system design is missing or incomplete. Moreover, the approach is able to derive a system description which can be used for model-based diagnosis. The approach was successfully tested on IPC-based robot control software like the rescue robot Lurker. IPC is a widely used basis for robot control software. Therefore, our approach is instantly usable on many different robot systems. Moreover, the approach can be used for model learning for any component-based software using an event-based publisher-subscriber mechanism for communication.

Currently, we are working on a port for Miro-based systems. This even will increase the number of potential target systems of our approach. Moreover, we work on the recognition of the behavior of additional connection types in order to enrich the generated models.

Moreover, we believe that the consideration of the content of the events will lead to significantly better models and diagnosis. For the modeling the techniques of Qualitative Reasoning seem to be promising. But it is an open question how such qualitative models can be automatically learned from recorded data.

References


Adding Fault Adaptive Control To Embedded Systems

Willibald Krenn and Franz Wotawa
Graz University of Technology
Institute for Software Technology
wkrenn,wotawa@ist.tugraz.at

Abstract
Fault adaptive control is an indispensable element in the design of robust devices. Often, however, the question arises, whether it is possible to add fault adaptive control to existing devices with minimal additional costs. We report our experiences learned when integrating self reasoning capabilities with an industry supplied prototype embedded system.

The self reasoning capabilities added to the device rely on a rule set that stores all possible behavior and includes information about unexpected faults, the number of times a certain rule was activated, and preferred behavior. At runtime the reasoning engine tries to find sequences of rules that transform the current system state to a goal state as specified in the rule set. We show how the rules that form the system model can be extracted from existing C code and discuss the immediate gain in robustness on the basis of a real hardware fault.

1 Introduction
"Survival of the fittest", due to Spencer, still drives our economy. Companies that need to prevail in this highly competitive environment are always searching for innovations so they can outclass their competition. One main area of separation thereby is product quality: Being able to offer the better product at a similar or even lower price is what companies are constantly trying to achieve.

For a company in the embedded systems domain, one possible piece in the puzzle is to offer products that feature a very high degree of durability and robustness. If this can be combined with maintainability and extendibility, the company has a good starting position to keep an edge over the competition.

Robustness is a platform attribute, so every part of the product design adds to the overall rating. For example, robust devices need a certain level of functional redundancy to compensate for permanent component faults. Taking the Artificial Intelligence (AI) aspect from the bouquet of necessary techniques, we quickly end at the problem of robust planning, diagnosis, and repair. These aspects are complemented by the business needs of preserving existing investments (i.e. existing code), low (better none) additional costs in order to use AI techniques, and, very important, a flat learning curve of newly introduced techniques.

We report of our experiences learned when integrating AI techniques, fostering robustness, with an industry supplied prototype embedded system. In particular the topics covered are

- We describe the runtime engine used to control the device.
- We present the structure of the original control program, written by an industry expert.
- We discuss how a model of the prototype was created from the original code.
- We compare the two firmware versions, showing that the runtime based version adds a significant level of robustness.
- Finally, we give a discussion of experiences learned.

The main contribution of this paper is twofold: First, we show how reasoning techniques can be introduced in an industry supplied product as evolutionary step. Evolutionary thereby means to preserve the original firmware as much as possible and to work with unchanged hardware. Second, we show how the reasoning technique improves the system’s robustness even in very simple scenarios. One of the presented scenarios thereby is a go/no-go one, meaning that the original device would have suffered a complete functionality failure, while the device with added reasoning techniques still provides most of the requested services. Finally, we add a discussion of the difficulties encountered and the experiences learned.

The paper is organized as follows: We start by presenting the prototype system that was provided by our industry partner and successively used to evaluate the integration of reasoning techniques. We need to give a brief overview of the system architecture and the most important hardware components, so that the implications of a component’s malfunction can be anticipated. After laying out the basic architecture, we introduce the chosen format of the system model and explain how the reasoning engine on board the device uses the supplied model to decide which action sequence to take. Having sketched the hardware requirements and the runtime environment (i.e. reasoning engine), we discuss the process of creating a suitable system model from the existing firmware. In
a next-to last section we evaluate the approach and discuss related research. Finally, we conclude the paper.

2 System Architecture

In this section we give an overview of both, the software and hardware architecture of the original device. In essence, we describe the starting point of our efforts to improve the device’s reliability by including self reasoning capabilities. The device, as sketched in this section, also serves as the sparring partner for our advanced version: In Section 5 we’ll compare the two versions of the device, which only differ in the presence of the reasoning engine.

The hardware design of the device is fairly straightforward and simple. Because the device has to act autonomously, one of the most important system components is a power module: An accumulator is fed with electrical energy drawn from a solar-cell array. At the same time, the accumulator also feeds the main system with electrical power. In order to react to dangerous battery conditions, some simple current monitoring circuit is present in the power module: In addition to measuring the current flowing in (or out) of the accumulator, the monitoring circuit also measures the accumulator voltage. Via I2C bus the power-monitor is connected to the main system.

The mentioned I2C bus also connects other peripheral components with the main controller: Most notably are a small display, a real time clock and an acceleration sensor.

While the power module is a very important subsystem, the heart of the device still is a Microchip PIC 18F microcontroller. The PIC runs the main control program that coordinates all hardware modules, determining the device’s overall behavior. This also means that the micro controller is the master on the I2C bus. Apart from the I2C bus, two UARTs provide a direct connection to a GSM and a GPS module respectively.

The main purpose of the device is to transmit information about the current position to some remote terminal: The preferred operation mode is to acquire position information by using the GPS-component and send it by issuing a GSM call. Relying on the GSM/GPS components alone, however, is dangerous because a hardware defect in one of them potentially renders the whole device useless. So an intelligent device should know about other (probably less preferred) ways how to transmit position information and choose among them according to the situation it is in. This brings us to the topic of the system software architecture.

Like any operating system, the firmware of the controller has to run forever. If written in C, as in our case, it is therefore reasonable to expect a while(true) loop in the main function. Indeed, when inspecting the source code, which was provided alongside the prototype, we immediately could identify the main function looking as expected. Within a loop, some flags are checked and the program flow branches accordingly. Conceptually, this code represents a big switch-case statement.

For the further discourse, we have to mention that the PIC micro-controller features two different priority levels of interrupts. Semantically, this gives the device three priority layers: If no interrupt occurs, the micro controller, simply speaking, runs the main function (program). If a low priority interrupt occurs, the controller interrupts the main program and continues execution at the low priority interrupt vector address. When servicing a low priority interrupt, or when running the main program, a high priority interrupt will interrupt the execution and the micro controller will continue execution at the high priority interrupt vector address. Clever use of priority management in connection with the timer interrupt gives the system programmer a cheap way of doing multi-programming.

Further inspection of the provided C code showed that the industry expert relied on exactly the discussed pattern to implement essentially two "threads" and an additional interrupt layer: The main program is responsible for the GSM communication, which, due to immanent timing constraints, has lowest priority. The timer interrupt periodically starts another task, thereby interrupting the main program. This second task essentially calculates the position information and does some I2C communication. Managing the UART buffers is done via high priority interrupts.

This already concludes the description of our prototype system. In the next section, we present our rule-set based approach to add self reasoning capabilities to the device.

3 Self Reasoning

Because we’re working with an autonomous, embedded device, we cannot afford storing an overly complex system model on-board: Reasoning about the system must be possible with bounded memory and bounded time. At the same time we need a model that is high-level enough to allow easy modification of behavior. In a first version, we settled for a rule based system, because it has a flat learning curve and it fits quite well to the hypothesis of the main program being a big-switch-case construct (see previous section). Moreover, extracting the conditions of this virtual switch statement for inclusion in the rule set can well be supported by automated techniques ([Wotawa and Krenn, 2007]).

We present a refined version of our system model that is more explicit and has an easier to understand semantics behind. The model is based on propositions that hold the current belief state of the system. Observations of the outside world are mapped by setting propositions true or false at fixed times. Besides these propositions, rules are a central element of our model: Rules interconnect different propositions and also describe valid action sequences the system can trigger in order to move the system to a valid goal state. Goals, thereby, are simply specially treated rules. Whenever there are several rules that “turn on” a proposition, we use a weight function to determine the precedence.

3.1 Rule-Base

We need to give a few definitions in order to present the semantics of the proposed rule based system. We start by defining the sets the system operates on:
Propositions \( P \)
Memory \( M \subseteq P \)
Labels \( L \)
Rules \( R \)
Actions \( A \)

\( P \) contains all possible propositions. If a proposition is believed to hold in the current world state then it is element of the memory, hence \( M \) is a subset of (or equal to) \( P \).

**Definition (Rule).** We define a rule as n-tuple:

\[
\text{Rule } r = (\text{label} : l \in L, \text{guard} : \bigwedge_{p_i \in P} p_i, \text{action} : \bigwedge_{a_i \in A} a_i, \text{postcond} : \{ \bigwedge_{\text{op}_i(p_i)} p_i \text{ occur only once, } \text{op}_1 \in \text{Ops}_1 \bigwedge_{\text{op}_2(l)} l_i \in L, \text{op}_2 \in \text{Ops}_2 \} \bigwedge \text{activity} : a \in [0, 1], \text{damping} : d \in [0, 1], \text{weight} : w \in [0, 1], \text{max} : m \in [0, 1], \text{profile} : \alpha : [0, 1] \mapsto [0, 1])
\]

\( \text{op}_1, \text{op}_2 \) denote an operation from the following sets:

\[
\text{Ops}_1 = \{\text{add}(p \in P) : M \mapsto M, \text{remove}(p \in P) : M \mapsto M\}
\]

\[
\text{Ops}_2 = \{\text{incweight}(l \in L) : [0, 1] \mapsto [0, 1], \text{decweight}(l \in L) : [0, 1] \mapsto [0, 1]\}
\]

Note that we restrict ourselves to post-conditions that only have one \textit{add} operation.

If the guard (including actions) of a rule holds, the system guarantees the state described by the post-condition. The activity factor (\textit{activity}) is a number representing how frequently the rule was chosen. The damping factor (\textit{damping}) indicates how frequently action predicates failed, when the guard was satisfied. Note that this situation should never happen. If it does, the runtime system will run a repair action that has to report success. The activity profile function \( \alpha \) is used for mapping the activity value \textit{activity of a rule to some value in the interval } \([0, 1]\). The \textit{max} value may be obtained from \( \alpha \): In that case it represents the input value which causes \( \alpha \) to return a maximum value.

**Definition (Activity Maximum).** As already said, \textit{max} can be calculated from \( \alpha \), in which case we introduce a function

\[
\text{Max} : ([0, 1] \mapsto [0, 1]) \mapsto [0, 1]
\]

that takes a function \( \alpha \) and returns the input value where \( \alpha \) becomes maximal. We need \textit{max} for weight calculation, as can be seen in the next definition.

**Definition (Weight Function).** Each rule gets assigned a weight which is calculated by some function

\[
\gamma : [0, 1] \times [0, 1] \times [0, 1] \times ([0, 1] \mapsto [0, 1]) \mapsto [0, 1]
\]

that calculates the weight as

\[
\gamma(\text{activity}, \text{damping}, \text{max}, \alpha) = \\
\alpha'(\text{activity}) \cdot \text{Abs(max }- \text{ activity}) \cdot (1 - \text{damping})
\]

We assume \textit{max} \( = \text{Max}(\alpha) \).

In order to be self contained, we briefly recapitulate the rationale behind the weight model in the next subsection of the paper. For more information we refer the interested reader to [Krenn and Wotawa, 2007b]. However, before we discuss the weight model in greater detail, we give the definition of the interpretation of a rule.

**Definition (Interpretation).** A rule \( r \) is run by some function \( I : M \times R \mapsto M \) defined as

\[
I(M, r \in R) = M' \text{ with }
\]

\[
M' = \begin{cases} 
M & \text{if guard}(r) \subseteq M \\
M \cup \{p|\text{add}(p) \in \text{postcond}(r)\} & \text{if guard}(r) \not\subseteq M \\
\{p|\text{remove}(p) \in \text{postcond}(r)\} & \land \text{action}(r) \\
\end{cases}
\]

and

\[
\text{weight}(x)' = \gamma(\text{weight}(x), \text{damping}(x), \text{profile}(x))
\]

and

\[
\text{weight}(x)' = \text{weight}(x) + c, \forall x \in R, l \in L : (\text{label}(x) == l \land \text{incweight}(l) \in \text{postcond}(r))
\]

and

\[
\text{weight}(x)' = \text{weight}(x) - c, \forall x \in R, l \in L : (\text{label}(x) == l \land \text{decweight}(l) \in \text{postcond}(r))
\]

Note that activity and damping factors, as outlined in the next section, get updated by the system too. The constant \( c \) stands for a user defined increment/decrement step.

When an action fails, \( M \) depends on the repair action function. To discuss the weight model in greater detail, we give the definition of the interpretation of a rule.

The power set of all rules \( Paths = 2^{\text{Rules}} \), if ordered and understood as set of sequences, gives all possible paths through the rule set. One path is a sequence of rules \( (r_1, r_2, ..., r_n) \). Running such a rule sequence means successive evaluation of each rule and, hence, a stepwise system state modification.

\[
I(M, \{r_1, r_2, ..., r_n\}) = I(I(M, r_1), \{r_2, ..., r_n\})
\]

\[
I(M, \{\}) = M
\]

Oversimplified, our reasoning engine constantly loops through all goals, sorted by weight, and searches for a sequence of rules that transforms the current system state into
the goal state, as specified by the guard of the goal. Whenever the system has to decide between two different rules that add the requested proposition to the memory, it uses a local-best strategy and tries the rule with the highest weight first.

Having said that, failures during action execution influence the future behavior of the device because failures increase the damping factor. The damping factor, in turn, is used to calculate the rule weight that is used to select between rules. The next subsection presents the weight model in more detail.

3.2 Weight Model

After presenting the semantics of the rule logic in the preceding section we need to give a more detailed explanation of the weight function. Weight calculation, which is also presented in [Kre1 and Wotawa, 2007b], is based on a quantitative model taking past experiences, activity, and user’s preferences into account. Thereby weight is calculated by \( \gamma(\cdots) \) and defined over the interval \([0\ldots1]\) with a saturating behavior. Following parameters are needed by \( \gamma \) for weight calculation: (A) The current activity \((\text{activity})\) of a rule, (B) experiences learned from past runs of the rule \((\text{damping})\), (C) some max-activity point \((\text{max})\) and (D) some user-supplied function \(\alpha(\text{activity})\) that takes the current activity of the rule as input parameter. As already mentioned in the previous section, we define:

\[
\gamma(\text{activity}, \text{damping}, \text{max}, \alpha) = \alpha'(\text{activity}) \cdot \text{Abs}(\text{max} - \text{activity}) \cdot (1 - \text{damping})
\]

where \(\alpha'(\text{activity})\) is the first derivative of \(\alpha(\text{activity})\). \(\alpha\) must not have arbitrary slope as \(\alpha'(\text{activity}) \cdot (\text{max} - \text{activity})\) has to be smaller than or equal to 1 in order for \(\gamma\) to stay within bounds. \(\text{Abs}(\text{max} - \text{activity})\) returns the distance to the user-defined maximum activity in a linear way. Finally, the last part of \(\gamma\) deals with experiences gained from past runs of the rule. It uses a damping factor \((\text{damping})\) which is a counter of failed attempts to execute the rule after all conditions within the rule were determined to be true. As already indicated, \(\text{damping}\) is defined on the range \([0\ldots1]\) and we say \(1 - \text{damping}\) to be the desirability.

Both, the damping and the activity factor of a rule get determined by the system during runtime in a predefined way. Function \(\alpha\) and the maximum activity \(\text{max}\) for a rule have to be provided by the rule designer and serve the purpose of influencing rule selection in two different ways: The maximum value \(\text{max}\) is used to specify how often a rule should be selected over a given time frame whereas \(\alpha\) is used to “correct”, i.e. boost, the weight over a specified (user-selected) activity range. Thereby \(\alpha\) helps deciding between different rules reaching the same goal in case \(\text{max}\) of both rules has the same value. (\(\alpha\) allows the developer to change the slope of \(\text{max} - \text{activity}\) over a certain range.) We do not especially demand for \(\alpha\) to have its maximum at the place of \(\text{max}\) but we demand a slope \(\geq 1\) until the specified \(\text{max}\). As already mentioned, the additional requirement is that \(\alpha'(\text{activity}) \cdot (\text{max} - \text{activity}) \leq 1\) in order for \(\gamma\) to stay within bounds. (As we also stated that \(\gamma\) is saturated, place-wise breach of this requirement won’t harm.)

After we have shown how to calculate the weight of a rule, we give the rationale behind. It is assumed that all rules stored within the knowledge base are considered to help the system advance to a “good state”. In other words, if the guard of a rule is satisfied it is beneficial for the system to choose the rule and run it\(^1\). The assumption is made here that after the guard is satisfied all actions possibly attached to the rule will carry out without error. The system monitors action execution and will increase the damping factor if it encounters an unexpected error while executing the selected rule. This way transient faults (and permanent ones) are masked from occurring too often. Note that the system also will decrease a damping factor, when a rule is run without failure and the damping factor of the rule is greater than zero.

We believe that most of the time the developer has different levels of activity in mind for different rules. In our system we use supplied \(\text{max}\) values to implement this behavior: Depending where the user puts the maximum, the system will run the associated rule with a given frequency. We want to emphasize that no rule can be totally blocked from execution in order to prevent the system to run out of run-able rules.

After sketching the system model, we discuss on basis of an example how to derive the system model from existing C code. For more rigorous discussion of the weight model, see [Kre1 and Wotawa, 2007b].

4 Bridging Two Worlds

One of the most important points in our work was to preserve the existing firmware as much as possible and to match the existing behavior of the system.

Figure 1 shows an excerpt of the original control program, namely the low-priority timer interrupt service routine. It can be seen that the supplied source code is well structured and that different behavior is encapsulated very well. Unfortunately, though, on closer inspection, we observed that in a few tasks that are called from the timer interrupt routine, global variables are decremented until zero. If a variable reaches zero, some calculation is performed and the global variable is

\(^1\)Provided that the rule is needed to reach a goal.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22-24, 2008, Blue Mountains, NSW, Australia.
again set to some value greater than zero: This pattern is used to stretch the timer frequency for a particular task. In order to use any runxTask function as action within the rule set, it is necessary to separate the timer code from the code doing the calculation. This is unfortunate, because it requires slight changes to the existing C code.

An even bigger problem than the one described is the fact that our rule based system, which was designed before we had access to the final version of the prototype, did not take the concept of multi-programming by using the timer interrupt into account. As such it is not designed to emulate the preemptive semantics that the interrupt offers. In fact the runtime system does not know anything about interrupts because a main idea of the design is that asynchronous events are signaled by adding a proposition to memory. As the runtime system is constantly trying to reach all goals it will - at some point in time - pick up the new situation and react accordingly. Of course we are aware of the fact that we cannot provide any hard real-time semantics that way but we figured that we did not need to provide any in the domain of tracking devices. That said, our runtime offers a more flexible way of "scheduling" the tasks than can be found in the original firmware.

Translating the code found in Figure 1 into a rule base is quite straightforward: Each task (except for the DBG and the GSM tasks) gets its own goal within the rule base. As the developer can use $\gamma$ and/or some proposition to tell the system how often the goal should be reached, we can emulate the timer-semantics found in the original version. As already indicated, it is not possible to use this design pattern for the GSM task: The reason is the use of concurrency as explained below.

Figure 2 shows the main function of the original firmware. As expected, we see some code doing the system initialization and a while(true) loop that keeps calling runM2M forever. As already mentioned the main program runs at lowest priority and is responsible for GSM communication to the remote data terminal. Inside the runM2M subroutine we find code like the following:

```c
while (m2mTimer && !getGSMEvent(GSM_EVENT_CONNECT));
```

The major point here is that this code - in order to work as expected - relies on a preemptive scheduler: The function getGSMEvent polls a flag set from the runGSMTask function. In addition, m2mTimer also is decremented by the timer interrupt service routine. Consequently, the call to runGSMTask has to occur while runM2M is running; otherwise, runM2M won't exit the shown while-loop. Because our rule base has to work with existing code, we cannot help but preserve the original behavior, which means that we cannot include runGSMTask in the rule set. (RunGSMTask is not re-entrant safe.)

So the first task when creating a system model from existing code is to determine which functions are to be included in the rule set. Due to several reasons (one shown) this cannot be done fully automatically. The second task then is to derive conditions under which these functions get called in the original code. In [Wotawa and Krenn, 2007] the authors propose a simple algorithm that helps deriving such conditions automatically. In addition, the proposed algorithm also provides

```c
while (1)
{
    initM2MTask();
    initSIMTask();
    initZIGTask();
    initACCTask();
    initRTCTask();
    initDISTask();
    initI2CTask();
    initGPSTask();
    initGSMTask();
    runGSMTask();
    printCRLF();
}
```

Figure 3 shows the simple rule set - omitting the timing information for clarity - we derived from the existing code. Without going into too much detail about the rule syntax, one can see that all functions found within the timer interrupt service routine (except for runDBGTask and runGSMTask) ended up being separate goals in the rule set. We did not include runM2M directly but instead two subroutines runM2M is calling: makeActiveCall and makePassiveCall add the propositions "makeActiveCall" and "makePassiveCall" respectively. One of these two is sufficient to satisfy the guard of the goal "goalDoGSMCom", making the system reach the goal.

Finally we end up with a very simple and intuitive rule set that, when presented to developers can be easily understood. In fact it is so simple that one might be tempted to ask what kind of advantage it brings? The first answer is a lot of flexibility and at the same time a concise high level description of what the system is supposed to do. What’s not so obvious is increased robustness, as we’ll show in the next section.

5 Evaluation and Discussion

As probably everyone can imagine, systems in a prototype stage have a life of their own. In our case, we were lucky because the system running the original firmware stopped work-
/*timer interrupt translates to:*/
goalGpsTask: <= Do(System.runGPSTask);
goalLedTask: <= Do(System.runLEDTask);
goalRtcTask: <= Do(System.runRTCtask);
goalAccTask: <= Do(System.runACCTask);
goalSigTask: <= Do(System.runSIGTask);
goalSimTask: <= Do(System.runSIMTask);
goalM2MTask: <= Do(System.runM2MTask);

/*run M2M translates to:*/
makePassiveCall
  <= !Test(GSM_EVENT_RING) & Do(System.makePassiveCall);
makeActiveCall
  <= Test(GSM_EVENT_RING) & Do(System.makeActiveCall);
goalDoGSMCom:
  <= Reach(makePassiveCall) | Reach(makeActiveCall);

Figure 3: Rule Set

Table 1: Needed Memory

<table>
<thead>
<tr>
<th></th>
<th>Prog. Memory</th>
<th>Data Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>original Firmware</td>
<td>8494 bytes</td>
<td>1528 bytes</td>
</tr>
<tr>
<td>model based FW</td>
<td>47996 bytes</td>
<td>3294 bytes</td>
</tr>
</tbody>
</table>

Adding Fault Adaptive Control To Embedded Systems

Table 1 compares the two different firmware versions in size. Because the PIC micro-controller uses a Harvard architecture, it has separate program and data memory. That said it is possible to use the program memory to store read-only data, so a good part of the additional memory used in the model based firmware is spent for representing the rule set. Note that our implementation is not optimized and that our firmware adds a real time operating system (RTOS) too. We need the RTOS in order to terminate actions after a timeout. Most of the added code and data comes from ready-to-use components or is automatically generated from the provided model. Areas left are:

- code necessary for interfacing the original C code
- code necessary for setting propositions in memory on external events
- repair functions that the runtime will call in case an action did not succeed

A more detailed overview of how much changes to the original source code were necessary provides Table 2. As can be seen, most of the changes are minor and related to the
Table 2: Necessary Code Changes

<table>
<thead>
<tr>
<th>File Name</th>
<th>+/- Lines</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc_task.c</td>
<td>4/1</td>
<td>timer interrupt decrement</td>
</tr>
<tr>
<td>dbg_task.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>dbg_uart.c</td>
<td>+/-</td>
<td>ext. changes due to RTOS</td>
</tr>
<tr>
<td>dis_task.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gps_data.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gps_fake.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gps_task.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gps_uart.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gsm_fake.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gsm_if.c</td>
<td>2/1</td>
<td>add GSM_EVENT_RING</td>
</tr>
<tr>
<td>gsm_task.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>gsm_uart.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>i2c_task.c</td>
<td>4/1</td>
<td>timer interrupt decrement</td>
</tr>
<tr>
<td>led_task.c</td>
<td>4/1</td>
<td>timer interrupt decrement</td>
</tr>
<tr>
<td>m2m.c</td>
<td>30/1</td>
<td>timer; add. extracted glue code</td>
</tr>
<tr>
<td>main.c</td>
<td>+/-</td>
<td>ext. changes (interrupt)</td>
</tr>
<tr>
<td>pwr_task.c</td>
<td>4/1</td>
<td>timer interrupt decrement</td>
</tr>
<tr>
<td>report.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>rtc_task.c</td>
<td>5/2</td>
<td>timer interrupt decrement</td>
</tr>
<tr>
<td>sim_task.c</td>
<td>0/0</td>
<td>identical</td>
</tr>
<tr>
<td>tim_task.c</td>
<td>0/1</td>
<td>deleted sync. with interrupt</td>
</tr>
<tr>
<td>zig_task.c</td>
<td>+/-</td>
<td>empty file. identical</td>
</tr>
</tbody>
</table>

timer interrupt decrement problem as discussed in Section 4. In addition following changes were made: In dbg_uart.c we added critical sections in order to avoid race conditions. To be able to add the event GSM_EVENT_RING to the world state, we changed the file gsm_if.c. Furthermore, we added automatically extracted glue code to m2m.c. The main.c file had to be modified in the area of interrupt handling, and we had to out-comment the main function. All files of the original firmware distribution (with listed modifications) are re-used in our intelligent runtime.

As can be seen, adding self reasoning capabilities to embedded devices increases computational cost. However, by having a declarative knowledge base and a safety net in form of a fault tolerant control layer, the developer does not have to think about fault tolerance when implementing low level functions which eases the developer’s task. Another property of the declarative knowledge base is that the developer can easily change the behavior of the device without having to touch a single line of C code. These facts combined with an easy to understand rule based model and a small execution layer make the proposed solution a very good candidate for adoption.

6 Related Research

Fault adaptive control is a major research area and therefore it has been used in a multitude of systems (e.g. [Williams and Nayak, 1997; Williams et al., 1998; Dearden et al., 2004; Biswas et al., 2003]). Since so much research has been done in that area, a lot of different techniques for robust planning, and robust execution have been proposed. Most relevant to our approach are [Saffiotti et al., 1995] and [Nilsson, 1994].

Multi-valued logic is used in [Saffiotti et al., 1995] to control intelligent agents. The authors also use weights to decide between alternatives. In particular, degrees of truth of formulas describing contextual (environmental) conditions are used to weight the preferences of different coordinated activities, e.g., following a corridor. The authors also introduce behaviors and goals and "relate behaviors to goals by defining the notion of goodness of a behavior for a goal." Behaviors are coordinated activities combined with contextual conditions and object descriptors. They can be combined and, thus, standard planning techniques can be used to generate combinations of behaviors to satisfy given goals. Our presented approach as well as [Saffiotti et al., 1995] use weights to select between alternatives. The biggest difference probably is that our approach wants to create a behavior (seen from outside) by combining the effects of running all stored (independent) goals from the knowledge base. Thereby \( \gamma \) determines the number of runs of a goal. In difference to desirability functions of [Saffiotti et al., 1995], \( \gamma \) is given by the system developer. The damping factor is a measure for desirability but it ignores the state the system is currently in and is inferred from the past.

Our approach to fault adaptive control can also be compared to TR-programs [Nilsson, 1994]. The biggest difference between our approach and TR-programs is in action selection and real-time capability: Our approach relies on \( \gamma \) functions and discrete time steps. It does not guarantee real time capabilities while TR-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 repair. 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 rule sets, we can compose a system that comprises several runtime systems.

7 Conclusion

We gave an overview of our approach to fault adaptive control in embedded systems. In particular we showed how to extend traditional embedded systems firmware with a self reasoning layer that selects actions based on information provided by a system model and experiences from past runs of an action.

In our experiments we showed that simple hardware faults can be handled in a smooth way preventing the whole system from being permanently inoperable. Hence, the proposed knowledge-based system, which uses a model of the available system’s actions and intentions, increases robustness. In contrast to the more traditional model-based diagnosis approach, our approach does not explicitly provide fault localization. Instead the approach provides means for automated re-configuration and function degradation depending on the available functional redundancy.
Acknowledgments
This work has been supported by the FIT-IT research project
SElf Properties In Autonomous Systems (SEPIAS) which is
funded by the Austrian Federal Ministry of Transport, Inno-
vation, and Technology, and FFG.

References

[Biswas et al., 2003] G. Biswas, G. Simon, G. Karsai, S. Ab-
delwahed, N. Mahadevan, T. Szemethy, J. Ramirez,
G. Péceli, and T. Kovácszály. Self-adaptive software
for fault-adaptive control. Third Intl. Workshop on Seld-

[Dearden et al., 2004] R. Dearden, F. Hutter, R. Simmons,
S. Thrun, V. Verma, and T. Willeke. Real-time fault de-
tection and situational awareness for rovers: Report on the
mars technology program task, 2004.

Wotawa. Collaboration of intelligent, autonomous sys-
tems: Situation aware behavior change. In Proceedings
of the International Symposium on Intelligent and Dis-

Wotawa. Gradient-based diagnosis. In Proceedings of the
International Workshop on Principles of Diagnosis, pages

agent control. Journal of Artificial Intelligence Research,

[Saffiotti et al., 1995] A. Saffiotti, K. Konolige, and E. H.
Ruspini. A multivalued-logic approach to integrating plan-
ning and control. Artificial Intelligence, 76(1-2):481–526,
1995.

[Williams and Nayak, 1997] Brian C. Williams and P. Pan-
durang Nayak. A reactive planner for a model-based ex-
ecutive. In M. Pollack, editor, Proceedings of the 15th
International Joint Conference on Artificial Intelligence

[Williams et al., 1998] Brian C. Williams, Nicola Muscet-
tola, P. Pandurang Nayak, and Barney Pell. Remote agent:
To boldly go where no AI system has gone before. Artifi-

Krenn. Knowledge extraction from c-code. In Workshop
on Intelligent Solutions in Embedded Systems, pages 49–
A Unified Information Criterion for Evaluating Probe and Test Selection

Juan Liu, Johan de Kleer, Lukas Kuhn, Bob Price, and Rong Zhou
Palo Alto Research Center
3333 Coyote Hill Rd, Palo Alto, CA94304, USA
{juan.liu, dekleer, lkuhn, bprice, rzhou}@parc.com
Contact: 650-812-4828, FAX: 650-812-4334

Abstract
Diagnostic tasks often need to make the decision of what measurement to make or what action to take in order to resolve ambiguities in diagnosis. Intuitively one would like to seek the most “informative” choice. In the paper, we formalize this intuition and propose an information criterion for evaluating and comparing measurement/action choices based on their information contribution. The criterion is mutual information, an information-theoretic concept measuring statistical dependence. The information criterion gives a precise quantitative metric to differentiate the quality of measurement/action choices. We use a few concrete example in two separate paradigms, probe selection in circuit diagnosis and test generation in production plants, to illustrate the mutual information criterion. Despite the apparent differences of the two paradigms, the information criterion works coherently. We demonstrate how different probing actions or test plans vary in their information values.

1 Introduction
A significant challenge in diagnostic tasks is to identify what new measurements to make next or what new experiments to try next in order to resolve ambiguities quickly. These two tasks are often termed “active probing” and “test generation”. While their goals are clear, the actual practice of selecting probing locations and designing test is more art than science. In this paper, we outline a general conceptual framework which unifies the tasks of selecting probing locations and test plans based on the concept of mutual information. The basic idea is simple: not all probing and test choices are equal; some are more informative than others. In this paper, we formalize the intuition using an information theoretic concept, mutual information. It provides a single metric to precisely evaluate the amount of information that a probing measurement or a test plan are expected to bring to the diagnostic problem. Using this metric, different choices can be compared fairly.

In this paper, we focus on the information evaluation criterion rather than the overall diagnosis. Figure 1 shows the general flowchart of sequential diagnosis. Based on a current diagnosis belief, one may decide where to make observations. This is the top block in the diagram marked as “probe selection or test generation”. The measurement is then taken, and used to update the belief via some inference mechanism such as a GDE (General Diagnostic Engine) [de Kleer and Williams, 1987] or a Bayesian inference engine [Berger, 1995]. With the updated belief, the probe selection/test generation process may repeat to find the next suitable action.

In this paper, we will not address the inference or the measurement process, but only focus on the probe selection/test generation part. Furthermore, this part involves an information evaluation criterion to measure the quality of different choices and a search strategy to find the optimal choice. The search is a sophisticated problem by itself. We will not address the search problem in this paper, but only show that the information criterion provides accurate heuristics to guide the search. The focus of this paper is on the information criterion (the block with thick border in Figure 1).

The information criterion for evaluating probe and test selections is most useful if active probing and test execution incurs a non-negligible cost. This is often the case as special equipments or technician labor are needed. On the other hand, if the probing and test execution is nearly effortless, then evaluation and selection would not be a problem. The burden is shifted to the inference part: being able to update the diagnosis in the presence of a large amount of test ev...
idence. In this paper, we restrict our discussion in the former case; furthermore, we assume that all probing actions or test plan executions incur a uniform cost. If the cost is non-uniform, we can devise the selection strategy to strike a balance between two competing goals: finding an action with the best information content, and yet keeping the action cost low. Action cost is typically known prior to taking the action, and our information criterion provides a metric for the information content.

In this paper, we first explain in Sec. 2, on an abstraction level, how the information criterion is formulated, computed, and used to guide the decision of which measurement to take. Then the information criterion is instantiated in two separate paradigms: (1) probe selection in circuit diagnosis, and (2) test plan generation in production plants. These two paradigms are conceptually different and have been addressed via quite different techniques, but we show that they share the common need for informative measurements, and can be unified under the information criterion. In Sec. 3, we show the information criterion for active probing in circuit diagnosis with two concrete examples: a simple linear cascade of inverter and a full adder. Our analysis shows that different probing locations vary a lot in terms of their information content, and the best probe location depends on the fault assumptions. In Sec. 4, we explain the information criterion for the plan selection in modular production plants. In this case, the information criterion takes a simple form and can be evaluated efficiently. In Sec. 5, we discuss a few possible extensions of the information criterion. The paper concludes with Sec. 6.

2 Mutual Information Criterion

To measure the information content, we consider mutual information, a concept rooted in information theory measuring statistical dependence. It is a commonly used metric for characterizing the performance of data compression and classification [Cover and Thomas, 1991].

For illustration, we use the following notation.\footnote{We use the standard notation, with upper case symbols denoting random variables and lower case symbols denoting a particular realization.} Let \( X \) be the underlying diagnostic state, for example, the bit-vector 011000 if the only second and third module have fault. Let \( Y \) be the observation, for instance, the measurement obtained at a probing location, or the outcome of a test plan execution. Note that \( X \) and \( Y \) are both random variables. The mutual information between \( X \) and \( Y \) is defined as an expectation:

\[
I(X;Y) = \sum_{x,y} p(x,y) \log \left( \frac{p(x,y)}{p(x)p(y)} \right). \tag{1}
\]

Conceptually, it measures the amount of information (in bits if logarithm is in base 2) the observation \( Y \) tells about the underlying diagnosis state \( X \). It is non-negative, and is equal to zero if and only if \( X \) and \( Y \) are independent, in which case, measuring any value of \( Y \) has no implication on refining the underlying diagnosis \( X \), hence has zero information content. In practice, we should avoid such an irrelevant observation.

But rather make an observation that reveals as much information as possible regarding \( X \).

In diagnostic tasks, observations are made from probing locations or test executions. Hence \( Y \) is actually implicitly parametrized by the probing location or the test plan. To emphasize this, we use \( Y_m \), with \( m \) denoting the measuring action. The goal for probe/plan selection is to find the \( m \) such that \( I(X;Y_m) \) is maximized. This involves search over all possible plans; we will not address the search problem in this paper. Rather, it is straightforward to compare probes/plans. For example, given two choices \( m_1 \) and \( m_2 \), we say \( m_1 \) is more informative and preferable than \( m_2 \) if

\[
I(X;Y_{m_1}) > I(X;Y_{m_2}). \tag{2}
\]

Mutual information admits an entropy interpretation. Entropy (and conditional entropy) has been an well-accepted and widely-used metric for uncertainty. For a random variable \( X \) with probability distribution \( p(x) \), its entropy is defined as \( H(X) = \sum_x p(x) \log \frac{1}{p(x)} \). The entropy measures the uncertainty in the random variable; the bigger, the more uncertain. It also serves as a bound for diagnosis task: to resolve ambiguities in a diagnosis problem with entropy of \( h \) bits, the number of tests we need on average is at least \( h \). Mutual information can be connected to entropy via the following form:

\[
I(X;Y) = H(X) - H(X|Y), \tag{3}
\]

where \( H(X) \) is the entropy of \( X \), and \( H(X|Y) \) is the entropy of \( X \) conditioning on observing \( Y \), i.e., the “remaining uncertainty” after the observation. Maximizing \( I(X;Y_m) \) is equivalent to minimizing \( H(X|Y_m) \). This is equivalent to say: we would like to select the best probe/plan, which leaves as little uncertainty as possible. This intuition of minimizing conditional entropy is used for example in the General Diagnostic Engine (GDE) [de Kleer and Williams, 1987].

To calculate mutual information, we take advantage of the symmetry of mutual information, i.e., \( I(X;Y) = I(Y;X) \). The amount of information that \( Y \) tells about \( X \) is equal to the amount that \( X \) tells about \( Y \). Exchanging \( X \) and \( Y \) in (3), we have

\[
I(X;Y) = H(Y) - H(Y|X). \tag{4}
\]

Although (3) and (4) are equivalent, the latter is often much easier to compute, since it uses the observation likelihood \( p(y|x) \) which is often known a priori. In contrast, (3) uses the posterior belief \( p(x|y) \), which is a lot harder to compute.

Mutual information has been used as an evaluation and selection criterion in a number of applications. For example, [Liu et al., 2003] uses mutual information to decide which sensors to activate in the context of tracking a moving target. Similarly, [Hoffmann et al., 2006] uses an information criterion to control a fleet of robots, sending robots to most advantageous locations. Medical researchers have used mutual information to guide feature selection to diagnose human lung cancer [Tourassi et al., 2001]. Related to machine diagnostics, [Verron et al., 2007] used a similar criterion in process diagnosis based on Bayesian networks. Its information criterion is derived for continuous health states. In this paper, we extend this framework to the discrete diagnostics domain.
and make connections between high-level intuitions and precise information evaluation.

3 Probe Selection

Circuit diagnosis is a canonical example of model-based reasoning. In this paradigm, a number of techniques have been proposed. The early GDE work [de Kleer and Williams, 1987] proposes a minimum entropy criterion that determines what measurements to make next: the one minimizing the conditional entropy of candidate probabilities resulting from the measurement. This is well aligned with our information criterion. In this section, we extend this early work to a variety of fault assumptions.

Here we use a simple example of a cascade of inverters (Figure 2). The same circuit was also used as a working example for illustration in [de Kleer and Williams, 1987]. An inverter gives $y_{\text{output}} = \neg y_{\text{input}}$ if it is working properly. An inverter that has a fault may produce wrong output. In the strong fault assumption, we assume that faulty inverter operates in a known way. Or one may retreat to a weak fault assumption, assuming that a faulty inverter operates in an unknown way and produces incorrect output for some inputs. In practice, intermittency of fault may add further complications. For instance, a faulty component may not always malfunction. Its malfunctioning is a random event with probability $q$ due to unknown environmental conditions. The diagnosis of intermittent fault has been addressed for example in [de Kleer, 2007]. In the analysis below, we consider four combinations: strong and weak faults, persistent ($q = 1$) and intermittent ($0 < q < 1$), and show how the fault assumption affects the optimal probing locations.

In this example, with four cascaded inverters, the underlying hypothesis space contains 16 hypotheses: $X = \{0000, 0001, \ldots, 1111\}$. Active probing in this case is to compare the probing locations $m = \{a, b, c, d, e\}$ and compute the respective mutual information values $I(X; Y_m)$.

3.1 Strong fault models, persistent

In the strong fault case, a faulty inverter operates in a known way, e.g., $y_{\text{output}} = \neg y_{\text{input}}$. In practice, the input/output relationship of a faulty component can be learned via diagnostic inference. To evaluate the mutual information, we first need to specify the observation likelihood model $p(y_m|x)$. Note that in this linear cascaded inverter example, we have,

$$p(y_m|x) = \sum_{(y_1,\ldots,y_{m-1},y_{m+1},\ldots,y_K)} p(y_1, y_2, \ldots, y_K|x),$$  \hfill (5)

where $K$ is the total number of components in the linear cascade ($K = 4$), and

$$p(y_1, y_2, \ldots, y_K|x) = p(y_1) \prod_i p(y_{i+1}|y_i, x_i)$$  \hfill (6)

The individual term $p(y_{i+1}|y_i, x_i)$ is the property of the $i$-th inverter module, with $y_i$ as its input and $y_{i+1}$ as the output. The variable $x_i$ is 0 if the $i$-th inverter has no fault and 1 otherwise.

The component-wise likelihood function is the following:

$$p(y_{i+1}|y_i, x_i) = \begin{cases} 1 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 0; \\ 0 & \text{if } y_{i+1} = y_i \text{ and } x_i = 0; \\ 1 & \text{if } y_{i+1} = y_i \text{ and } x_i = 1; \\ 0 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 1; \end{cases}$$  \hfill (7)

Plugging the component-wise likelihood into (6) to get the joint distribution $p(a, y_2, \ldots, y_K|x)$ and marginalizing (as in (5)) to obtain $p(y_m|x)$, we can evaluate the mutual information.

Under the strong fault assumption, given any hypothesis $x$ and initial input at point $a$, there is no uncertainty in the output of any inverter, hence the second term $H(Y_m|X) = 0$. Therefore, we simply have $I(X; Y_m) = H(Y_m)$.

To illustrate the information criterion, let us consider a concrete example. For the cascaded inverters show in Figure 2, with input $a = 1$ and output $e = 0$, there must be something wrong with this circuit, and the diagnostic task needs to decide which location to probe. The mutual information $I(X; Y_m)$ for different probing locations $m = \{a, b, c, d, e\}$ is listed in Table 1 (second column) under the strong persistent fault assumption. The initial condition is that the inverters $A, B, C, D$ are independently faulty with probability 0.2, 0.1, 0.1, 0.1 respectively. The best probing location in this case is $b$, immediately after the first inverter. This is intuitive, given that inverter $A$ is most likely to have fault than others. The two ends $a$ and $e$ has zero information value, since we already know their values $a = 1$ and $e = 0$.

The preceding likelihood function (7) was derived by direct inspection. Model-based diagnosis algorithms such as GDE compute $p(y_m|x)$ through first principles reasoning from a description of the system. For example, the 1st line of equation (7) is inferred from the fact that a correctly working inverter ($x_i = 0$) always $(p(y_{i+1}|y_i, x_i = 0) \text{ inverts its output } (y_{i+1} = \neg y_i).$ For more details of such algorithms see [de Kleer and Williams, 1987].
3.2 Strong fault models, intermittent

The component-wise likelihood function is:

\[
p(y_{i+1}|y_i, x_i) = \begin{cases} 
1 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 0; \\
0 & \text{if } y_{i+1} = y_i \text{ and } x_i = 0; \\
q & \text{if } y_{i+1} = y_i \text{ and } x_i = 1; \\
(1 - q) & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 1; 
\end{cases}
\]  

(8)

To evaluate mutual information, we need to compute \( p(y_{m}|x) \). Starting from point \( a \), the probability of \( p(y_{i+1}|x) \) can be evaluated recursively from \( p(y_i|x) \), until reaching the probe location \( m \). We use the shorthand notation \( p_i(b) \) for \( p(y_i = b|x) \) with \( b = 0, 1 \). The evaluation is recursive:

\[
p_{i+1}(b) = \begin{cases} 
p_i(\neg b) \cdot q + p_i(b) \cdot (1 - q) & \text{if } x_i = 1; \\
p_i(b) & \text{if } x_i = 0 \end{cases}
\]  

(9)

Through the recursion, we can push to the probe location \( m \) to evaluate the outcome probability \( p(y_m = b|x) \), and then compute the mutual information \( I(X; Y_m) \) as in (4).

The analysis above does not assume that all inverters have the same intermittent parameter \( q \). On the other hand, if they have the same \( q \) value, it further simplifies into

\[
p(y_m|x) = \begin{cases} 
\sum_{\text{even } k} C_N^k q^k (1 - q)^{N-k} & \text{for } y_m = a; \\
\sum_{\text{odd } k} C_N^k q^k (1 - q)^{N-k} & \text{for } y_m \neq a
\end{cases}
\]

Here \( N \) is the total number of modules that have fault before the probing location \( m \) in the hypothesis \( x \), i.e., \( N = \sum_{i=1}^{m} x_i \). This result is also easy to understand: if the outcome at probing location \( m \) is the same as \( a \), then there must be an even number of malfunctioning inverters between \( a \) and \( m \), which can be chosen randomly from a total of \( N \) possible faults. Hence the probability has the \( C_N^k \) term and the polynomial term with even \( k \). Same for the case of \( y_m \neq a \), which must have an odd number of faults. From \( p(y_m|x) \), we can evaluate \( H(Y_m|X) \).

The third column of Table 1 shows the comparison of probing locations under the intermittent strong fault assumption. The best probing location is \( c \) instead of \( b \) as in the previous two cases. On the conceptual level, the output at a good probing location should have a decent probability of observing an actual malfunctioning unit. If that probability is too low, one cannot learn much from the measurement. In the intermittent fault case, the small \( q \) value means that the malfunctioning is rarely occurring. This causes the best probing location to shift towards the middle. In this example, if we observe \( c = 0 \), then there must be a fault in the first two inverters; if \( c = 1 \), then the last two inverters must have a fault.

3.3 Weak fault models, persistent

In the weak fault model, the observation likelihood is the following:

\[
p(y_{i+1}|y_i, x_i) = \begin{cases} 
1 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 0; \\
0 & \text{if } y_{i+1} = y_i \text{ and } x_i = 0; \\
0.5 & \text{if } y_{i+1} = 0 \text{ and } x_i = 1; \\
0.5 & \text{if } y_{i+1} = 1 \text{ and } x_i = 1; 
\end{cases}
\]  

(11)

In this case, the mutual information criterion is the following:

\[
I(X; Y_m) = H(Y_m) - p_{m|y} \]  

(12)

where \( p_{m|y} \) denotes the probability that the outcome at \( m \) is unknown. The derivation is straightforward: at any given measuring location \( m \), with any particular hypothesis \( x \in X \), its outcome can be 0, 1, or unknown. For instance, with \( a = 1, e = 0 \), the no-fault hypothesis \{000\} is ruled out. With the remaining hypotheses, the measurement at location \( b \) has a few possible values: (i) 0 under the hypotheses \( X_0 = \{0001, 0010, 0011, 0100, 0101, 0110, 0111\} \), since the first module has no fault; (ii) 1 under the hypothesis \( X_1 = 1000 \) since the last three modules have no fault; and (iii) unknown under all other remaining hypotheses \( X_{id} \). The conditional entropy

\[
H(Y|X) = \sum_{x \in X} H(Y|X = x)p(x)
\]  

(13)

This sum can be broken down into three sets: over the set \( X_1 \), the conditional entropy is 0 since the outcome is deterministic with value 1; same for the set \( X_0 \). The only remaining set is \( X_{id} \), in which each hypothesis \( x \) has a corresponding conditional entropy \( H(Y|X = x) = 1 \) bit from the equal probability 0/1 outcome, and the whole set has an accumulated probability of \( p_{m|y} \). Putting them altogether, we have (12).

The fourth column of Table 1 shows the \( I(X; Y_m) \) values. Similar to the persistent strong fault case, the best probing location is \( b \).

The prior work [de Kleer and Williams, 1987] proposes a minimum entropy criterion of selecting \( m \) to minimize \( H(X|Y) \). Through a lengthy derivation, its entropy criterion is \( -H(Y_m) + p_{m|y} \log M \), where \( M \) is the number of distinct values that \( Y_m \) can take. This is exactly the same as in (12). The mutual information derivation is much simpler, and can be readily generalized to a variety of fault assumptions.

3.4 Weak fault models, intermittent

For the weak fault, intermittency will make the output less random. The component-wise likelihood function is:

\[
p(y_{i+1}|y_i, x_i) = \begin{cases} 
1 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 0; \\
0 & \text{if } y_{i+1} = y_i \text{ and } x_i = 0; \\
q/2 & \text{if } y_{i+1} = y_i \text{ and } x_i = 1; \\
1 - q/2 & \text{if } y_{i+1} = \neg y_i \text{ and } x_i = 1; 
\end{cases}
\]  

(14)

The \( q/2 \) in the third line comes from the fact that a faulty module malfunctions with a probability of \( q \) and in that situation, the probability of observing a wrong output is 0.5 due to the weak fault assumption. In this particular cascaded inverter example, the intermittent weak fault case is identical to the strong intermittent fault case, except with a new \( q \) value. The mutual information criterion can be computed in the same way.

The last column of Table 1 enumerates the mutual information values under the intermittent weak fault assumption. The best probe location is \( c \). Another thing to note in Table 1 is that the information content decreases when the faults become intermittent. The persistent strong fault column has the highest values. This is because the observation likelihood model is very informative: observing the input and output of any inverter, one can immediately say whether the inverter has fault. In the intermittent weak fault assumption, the input-output observation is hardly conclusive: observing a correct
Table 2: The mutual information $I(X;Y_m)$ for different probing locations $\{x, y, z\}$ in the full adder circuit.

<table>
<thead>
<tr>
<th></th>
<th>Persistent strong</th>
<th>Intermittent strong $q = 0.1$</th>
<th>Persistent weak</th>
<th>Intermittent weak $q = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0.773</td>
<td>0.049</td>
<td>0.368</td>
<td>0.042</td>
</tr>
<tr>
<td>$y$</td>
<td>0.440</td>
<td>0.003</td>
<td>0.100</td>
<td>0.001</td>
</tr>
<tr>
<td>$z$</td>
<td>0</td>
<td>0</td>
<td>0.889</td>
<td>0.808</td>
</tr>
</tbody>
</table>

input-output pair does not mean the inverter is good. One can conclude the inverter is bad only when a malfunctioning shows up and the output happens to be wrong, but this is a very rare event by the intermittency nature. Hence we see this information value decreasing as we retreat to weaker and weaker assumptions.

3.5 Another example: full adder

Another example is the diagnosis of a full adder, shown in Figure 3. It exhibits a level of sophistication lacking in the previous example, but typical in real circuits: the gates are diverse, and the connections are non-linear. We show that the mutual information criterion can be used to guide the probe selection in this representative circuit.

The full adder takes three inputs: $a$, $b$, $c_i$ (carry-in), and produces two outputs $q$ and $c_o$ (carry-out). With inputs $\{a = 1, b = 1, c_i = 0\}$ and outputs $\{q = 1, c_o = 1\}$, it is clear that the circuit is incorrect and needs diagnosis. The possible probe locations are $x$, $y$, and $z$. Which one to probe is the choice that need to be made. We again consider four fault combinations. For the strong fault model, we assume the following: (1) any faulty XOR gate operates like an OR gate; (2) any faulty AND gate operates like a NAND gate; and (3) any faulty OR gate operates like a NOR gate. These are assumptions for illustration in this paper, and should be adjusted accordingly in real diagnosis problems. For the weak fault model, we assume any malfunctioning gate produces 0 or 1 with equal probability.

Table 2 lists the results under a variety of fault assumptions. Notice a few things from the table:

- Not all probing locations are equal, and some could even be useless. For instance, probing the location $z$ under the strong fault model (the second column, the last row) has zero information. Due to the strong fault assumption, only a handful of fault combinations could have produced the observed input/output relationship. Concatenating the gates in the order $(X1, A1, X2, A2, O1)$, the only possible fault combinations are: $X = \{10000, 10010, 10100, 10110, 11001, 11010, 11101, 11110\}$. Under all these combinations, we will have $z = 1$. Hence probing at $z$ does not contribute to the diagnosis.

- The best probe location changes as the fault assumption varies. For instance, the best probe location under the strong fault model (the second and third columns) are $x$, while the best probe location under the weak fault model (the last two columns) is $z$.

- With fault intermittency, the probing action becomes less informative. This is consistent with what we observed in the cascaded inverters example.

3.6 Extension: test vector generation

One diagnosis strategy is to choose suitable test vectors. By varying the input to the circuits, for instance, the input $\{a, b, c\}$ to the full adder, one may isolate faults and help the overall diagnostic task. Similar information criteria can be extended to test vector selection: to choose the most informative test vector. The same mutual information evaluation mechanism can be used. Instead of evaluating $I(X;Y_m)$ and choosing the best probe location $m$, we can change the variable $m$ to be the input test vectors, or even the combination of probe locations and test vectors. The detailed evaluation would be different, but the general idea of using mutual information to differentiate the quality of test choices remains the same. One direct approach to find the next best test vector is to apply GDE’s approach to every possible input vector.

4 Plan Selection

A common diagnostic problem is the diagnosis of a production plant. A product often goes through many steps or modules in the manufacturing process. When the outcome is unsatisfactory, one needs to diagnose which step or steps have caused the problem. In this section, we use PARC’s prototype modular printer (shown in Figure 4) for illustration. This printer has over 170 independently controlled modules and many possible paper paths; the redundancy enables high-speed high-throughput printing [Ruml et al., 2005]. The product in this case is a paper sheet, which enters from the left and exits on the right. It may go through paper path modules (dark black edges with small rollers in the figure) and printer modules (the 4 large rectangles). At the output, we may observe a fault; the most commonly observed is a damaged paper (wrinkled, ripped, or dog-eared). Unlike active probing in circuit diagnosis, we cannot make observations at arbitrary modules before a paper sheet exits. What can be leveraged is to control which modules the paper sheet goes through. For example, if we suspect the top-right printer module to have a fault but not the other three printer modules, we may control the path so that (1) it avoids the top-right printer if we want to maintain a working system, or (2) it passes the top-right printer, if our goal is to diagnose the suspected fault.
Although we use the prototype printer as our illustration vehicle, the analysis in this section is general and can be extended to the diagnosis of many production plants. At the abstraction level, the problem of plan selection is to choose the most informative production plan $P$ to maximize the information content $I(X; Y_P)$, where $X$ is the underlying diagnosis state, and $Y_P$ is the outcome of the production plan, which is binary-valued: 0 for success plan, and 1 for unsuccessful (e.g., damaged paper sheet).

### 4.1 Single intermittent fault

In practical situations, the number of potential faults is small. A simplification of the diagnostic problem is to assume that the whole system has at most one fault, known as the single fault assumption. It reduces the diagnosis space from exponential $2^M$ to $M$, where $M$ is the total number of modules, i.e., $X = \{1, 2, \ldots, M\}$. We further assume that if a module $i$ has fault, it has an intermittency probability of $q_i$.

The question is how to diagnose this modular printer when a fault has been observed? A common divide-and-conquer scheme is to devise a plan $P$ to go through only half of the modules. If the plan observes a fault, that means $P$ contains the fault, and the other half that $P$ excludes is cleared of suspicion. If the plan is successful, then $P$ is cleared, and the fault must be in the other half. In this way, every plan dissects the diagnosis space by half. This divide-and-conquer strategy is used for example in [Kuhn et al., 2008].

The strategy can be generalized via the mutual information criterion. In the production plant abstraction, each faulty module can damage the product with an intermittent probability $q_i$ if it is included in the production plan $P$, i.e., with the observation likelihood:

$$p(y_P | X = i) = \begin{cases} 0 & \text{if } y = 1 \text{ and } i \notin P; \\ q_i & \text{if } y = 0 \text{ and } i \notin P; \\ 1 - q_i & \text{if } y = 1 \text{ and } i \in P; \\ 1 & \text{if } y = 0 \text{ and } i \in P; \end{cases}$$

Define $H_q^{(i)}$ as the entropy corresponding to the binomial distribution $q_i$, i.e., $H_q^{(i)} \triangleq -[q_i \log q_i + (1-q_i) \log(1-q_i)]$. The mutual information can be evaluated in the following form

$$I(X; Y_P) = [-y_0 \log y_0 - y_1 \log y_1] - \sum_{i \in P} p_i H_q^{(i)},$$

where $y_0$ is the probability of observing a success, $y_1$ is the probability of observing a failure. The derivation follows from (4). The first term (the bracketed term) is $H(Y_P)$, and one can easily verify the second term is $H(Y_P | X) = \sum_{i \in P} p_i H_q^{(i)}$.

An interesting special case is when all faults are persistent, i.e., $q_i = 1$ for all $i$. In this case, all $H_q^{(i)} = 0$, and the second term in (16) vanishes. The mutual information is hence only $-y_0 \log y_0 - y_1 \log y_1$, maximized when $y_0 = y_1 = 0.5$. This means the path $P$ should go through half of the probability mass, i.e., $\sum_{i \in P} p_i = 0.5$. This is a generalization of the divide-and-conquer strategy above.

In the intermittent fault case, the second term is non-zero and can be considered as a "correction" term due to the intermittency. When all the modules have the same $q_i$ value (this is likely in the paper path modules which all have the identical design), the mutual information can be further simplified. It can be evaluated as the function of a single variable $w = \sum_{i \in P} p_i$:

$$I(X; Y_P) = -[w q \log w + (1-w q) \log(1-w q)] - w H_q,$$

Note it is very easy to evaluate: given any plan $P$, we can obtain $w$ as the summation of the probability mass along the plan, then plug in to obtain the corresponding mutual information value. The computation only involves several additions and multiplications.

Now consider the more complicated problem of plan generation. The optimal plan should maximize $I(X; Y_P)$. Since $w$ is the only variable, the optimal value of $w$ can be derived from simple calculus, and is

$$w = \frac{1}{q(2^M/2+1)}$$

When $q \rightarrow 0$, $w$ is asymptotically approaching $\frac{1}{2}$. For $q \in (0, 1]$, $w$ takes value from $\frac{1}{2}$ to $\frac{1}{2}$. This is an interesting result: as the faults become less likely to show, we should include less probability mass in the plan $P$. If the plan comes out with a damaged product, the rest of the probability mass not included in $P$ is ruled out by the single fault assumption. On the other hand, if the plan comes out without showing any damage, the modules in $P$ cannot be ruled out due to intermittency, but have to be tested further.

**Implication to the search:** The original path search problem, i.e., searching for the best path $P$ which has the maximal information gain, has been reduced to a much simpler yet equivalent problem of searching for a path $P$, which has an accumulative probability closest to a target value $w$. The latter problem is simple because the accumulative probability of the path is additive by nature. Handling a cost function that is additive helps the search problem tremendously, because (1) the order of modules in the path does not matter, (2) sub-paths
can be summarized from their contribution to the overall cost function, and (3) it enables tree pruning.

Given the target value \( w \) as in (18), efficient search algorithms can be used to find the best plan. The target-value search strategy proposed in [Kuhn et al., 2008] tackles this problem. It starts from the product entrance (or exit), grows the search tree, and prunes it by establishing upper- and lower-bounds on the deviation of the accumulated probability to the target value \( w \). The search is very efficient. Interested readers may refer to [Kuhn et al., 2008] for details.

4.2 Multiple faults

We extend the analysis to multiple faults. For simplicity, we assume all modules have identical \( q \) values. The diagnosis space is \( x = (0/1, \ldots, 0/1) \), where the \( i \)-th element value is an indicator function regarding whether this module has fault. For each possible diagnosis, we have a probability \( p(x) \).

To evaluate the mutual information \( I(X;Y) = H(Y) - H(Y|X) \), we compute the two terms as follows.

- The first term \( H(Y) \) is the entropy corresponding to the binomial distribution \( (y_0, y_1, \ldots, y_n) \), with
  \[
  y_0 = p(y = 0) = \sum_x p(y = 0|x)p(x) \tag{19}
  \]
  \[
  y_1 = \sum_x (1 - q)^k(x,P)p(x) \tag{20}
  \]
  where \( k(x, P) \) is the number of faulty modules that \( P \) goes through. \( k(x, P) = \sum_{i \in P} x_i \), \( k \) is a random variable with a distribution \( p(k) \). The distribution can be derived from the diagnosis belief \( p(x) \) and the plan \( P \). With \( p(k) \) computed, we have \( y_0 = \sum_k (1 - q)^k p(k) \).

- For any given value of \( k \), the outcome is 0 with probability \( (1 - q)^k \) and 1 with probability \( 1 - (1 - q)^k \), and we use \( H_k \) to denote the entropy corresponding to this distribution. The conditional entropy is
  \[
  H(Y|X) = H(Y|K) = \sum_k p(k)H_k \tag{21}
  \]

A potential path \( P \) affects the cost function only over \( k \) and its distribution, i.e., the number of faulty modules it passes. If two different paths offer the same distribution of \( k \), they are essentially the same from the mutual information perspective. Given any plan \( P \), we can evaluate \( p(k) \) and \( H_k \).

Note the following a few special cases:

- If \( q = 1 \), the second term is 0, and the optimal for the first term is \( y_0 = y_1 = 0.5 \) (selecting a path which has equal chance of observing a damaged/undamaged paper). This is the same as \( p(k = 0) = 0.5 \), i.e., with a probability of one half, all the modules in \( P \) are good.

- Consider the initial condition that all modules are independently faulty with probability \( s \) (\( s < 1 \)). In this case, we can choose the optimal plan length(\( M \)), and which ones to include in the path does not matter since all modules are identical in their faulty probability. With \( s = 0.5 \), the optimal path \( P \) goes through only one module. With smaller \( s \) values, the optimal length is longer.

To see this, note that \( p(k = 0) = C^n_k q^k (1 - q)^{n-k} \), and \( y_0 = (1 - q^s)^M \). For \( y_0 \) to get close to 0.5, \( M \) increases as \( s \) decreases.

The mutual information criterion can be used to guide the search for optimal production path. The exact search problem is difficult (the number of possible paths is exponential) and is out of the scope of this paper. However, the evaluation of mutual information is easy. This evaluation can be used to compare a few plan candidates, or make local adjustment to an existing plan — e.g., adding a new module or deleting an existing module, in order to obtain an informative measurement. This serves the diagnostics goal and minimizes the number of further tests.

5 Discussion

The two paradigms illustrated above, probe selection in circuit diagnosis and test plan generation in production plant diagnosis, both use a greedy strategy. At any step, the selection process uses the information criterion to find the most informative measurement to make for the time being. This greedy strategy works well for the diagnosis of static system, where the underlying ground truth of component fault does not change over time. However, there is no guarantee of optimality. On the other hand, the information criterion, \( I(X;Y_m) \) can be re-formulated with a look-ahead horizon, i.e., instead of computing the mutual information between \( X \) and the immediate probing action \( Y_m \), we can compute the mutual information \( I(X;Y_{(t=1)}, Y_{(t=2)}, \ldots, Y_{(t=T)}) \), where \( T \) is the look-ahead horizon. Using this criterion, one would be able to compare choices on their relatively long term contribution. The drawback is that the new criterion \( I(X;Y_{(t=1)}, Y_{(t=2)}, \ldots, Y_{(t=T)}) \) is much harder to evaluate. The state space grows exponentially. Various approximation techniques can be used, see [Hoffmann et al., 2006] for an example. Another strategy is to use greedy strategy most of the time, but switch to the look-ahead strategy only occasionally to avoid getting trapped in local optimum.

The search for optimal test sequence is known to be NP-hard. The work in [Tu and Pattipati, 2003] proposes a roll-out algorithm, inspired by policy iteration of dynamic programming, to search for a suboptimal solution. Our information criterion with single and multi-step lookahead can be readily combined with this roll-out strategy.

6 Conclusion

This paper proposes an information criterion for evaluating and selecting which measurement to make to help diagnosis. The criterion is based on mutual information, rooted in information theory to measure the dependence of random variables. The information criterion can be used in a variety of diagnostic problems, such as active probing in troubleshooting circuits and test plan generation in production plant diagnosis. From the analysis we can see that different action choices vary in their information contribution, and thus it is essential to be able to evaluate and compare them. The information criterion can further guide the search for optimal actions.
References


The Role of Agent Diagnosis in Multi-Agent Plan Repair

Roberto Micalizio  Pietro Torasso
Universit` a di Torino
Dipartimento di Informatica
{micalizio, torasso}@di.unito.it

Abstract

The paper addresses the problem of supervising the execution of a multi-agent plan (MAP), where actions are executed concurrently by a team of cooperating agents in a partially observable environment. Because of the cooperation among agents, the failure of the action of an agent may impact also the execution of the plans assigned to other agents. The paper formulates a distributed approach to agent diagnosis where each agent monitors its own sub-plan to detect action failures and explains these action failures in terms of faults in the functionality of the agent.

The paper discusses a methodology (based on agent diagnosis) for reacting to the action failure; first, MAP is revised in order to limit the harmful effects of the detected action failure; and second, a propagation step is performed to estimate the impact of the failure and to determine the missing goals (the sub-goals no longer achievable as a consequence of the failure itself).

1 Introduction

Within the AI community there is a growing interest in the development of autonomous systems; i.e., systems which are able to maintain themselves alive by properly reacting when unexpected events occur.

In general the autonomy is achieved by establishing a closed loop of control feedback (control loop in short), which typically involves many tasks such as on-line monitoring and diagnosing the system, but also (re)planning and (re)scheduling its activities and/or (re)configuring its components. Of course, the complexity and the relevance of the tasks included within a control loop may depend on the characteristics of the specific system under consideration.

The issues for establishing a control loop have found appropriate solutions in the scenarios where a single agent behaves as supervisor of the whole system (a significant example is the Livingstone architecture adopted in the Remote Agent [Muscettola et al., 1998]); on the contrary, the problem of the control loop in a multi-agent system has not found yet a general solution. Part of the difficulties lie in the large variety of possible organizations in multi agent systems. For this reason, we focus our attention on the class of distributed systems which can be modeled as a multi-agent plan (MAP); i.e., systems where agents cooperate in order to reach a common goal by executing actions concurrently. Recently a number of approaches to the synthesis of MAPs have been proposed (see e.g., [Boutilier and Brafman, 2001; Jensen and Veloso, 2000; Cox et al., 2005]); however, the synthesis of a MAP is just the first step, in fact the actual execution of a plan may be threatened [Birnbaum et al., 1990] by the occurrence of unexpected events (e.g., faults in the functionalities of the agents); this means that the execution of the MAP needs to be controlled in order to detect anomalous situations and to recover from these situations.

So far, some solutions for supervising (monitoring and diagnosing) the execution of a multi-agent plan have been proposed. The approaches that are more relevant for the present paper stem mainly propose distributed solutions where each agent is responsible for supervising the actions it executes.(see e.g., [Wittenveen et al., 2005; Kalech and Kaminka, 2007; Micalizio and Torasso, 2007a]) In these works, the task of diagnosis aims at detecting and explaining the anomalies which arise during the execution of actions. These anomalies may consist in disagreements among the agents in the team (see [Kalech and Kaminka, 2007]) or in the absence of expected effects after the execution of some actions (see [Wittenveen et al., 2005; Micalizio and Torasso, 2007a]). In [Kalech and Kaminka, 2007], the diagnosis singles out the root causes of the disagreement among the agents; while in [Wittenveen et al., 2005; Micalizio and Torasso, 2007a] the notion of plan diagnosis consists in a subset of actions whose failure is consistent with the anomalous observed behavior.

While these works represent a first attempt to integrate the steps of plan execution and diagnosis, in this paper we discuss a framework for also including a plan repair strategy. In fact, plan repair is fundamental for properly reacting to the action failures detected by plan diagnosis. The paper proposes a local repair strategy where an agent in trouble reorganizes its own activities to limit the impact of a failure in the global plan by releasing all the resources it holds; so all these resources are made available again to other teammates.

2 Modeling the Multi-Agent Plan.

In this paper we consider a specific class of Multi Agent Systems where a team T of “benevolent” agents cooperate to
reach a common complex goal \( G \). In particular, the global goal \( G \) is decomposed into a set of (easier) sub-goals; each agent in \( T \) has to reach one or more of these sub-goals. In most cases, however, the sub-goals are not independent of one another; in fact the agents cooperate by exchanging services and this cooperative behavior introduces causal dependencies among their activities. Thereby, when an unexpected event causes the failure of an agent activity, this failure may propagate in the whole system affecting the activities of the other agents in the team.

**Global plan.** The global task of the agents’ team can be properly represented via the notion of Multi-Agent Plan as formalized by Cox et al. in [Cox et al., 2005].

Given a team \( T \) of agents, the MAP is the tuple \( \langle A, E, CL, CC, NC \rangle \) such that:

- \( A \) is the set of the action instances the agents have to execute; each action \( a \) is assigned to a specific agent \( i \) of the team \( T \), and it is modeled in terms of preconditions and direct effects. Within the set \( A \) there are two special actions: \( a_0 \) and \( a_0^\prime \); \( a_0 \) is the starting action, it has no preconditions and its effects specify which propositions are true in the initial state; \( a_0^\prime \) is the ending action, it has no effects and its preconditions specify the propositions which must hold in the final state i.e., the preconditions of \( a_0^\prime \) specify the MAP’s goal \( G \).

- \( E \) is a set of precedence links between actions: a precedence link \( a \prec a' \) in \( E \) indicates that the action \( a \) must precede the execution of the action \( a' \);

- \( CL \) is a set of causal links of the form \( l : a \xrightarrow{\text{res}} a' \); the link \( l \) states that the action \( a \) provides the action \( a' \) with the service \( q \), where \( q \) is an atom occurring in the preconditions of \( a' \);

- \( CC \) and \( NC \) are respectively concurrency and non-concurrency symmetric relations over action instances in \( A \). The pair \( \langle a, a' \rangle \in NC \) is equivalent to the statement \( (a \prec a') \lor (a' \prec a) \); whereas, the pair \( \langle a, a' \rangle \in CC \) means that the two actions \( a \) and \( a' \) must be executed simultaneously; i.e., the pair \( \langle a, a' \rangle \) in \( CC \) models a joint action.

For sake of simplicity, in the present paper we do not consider joint actions, while the non-concurrency constraints are replaced by a suitable set of precedence and causal links: according to the concurrency requirement introduced in [Wittenveen et al., 2005], the conflicts for accessing the resources are solved during the planning phase by means of causal links of the form \( l : a \xrightarrow{\text{free(res)}} a' \) indicating that resource \( res \) is relinquished by agent \( i \) (responsible for action \( a \)) and assigned to agent \( j \) (which has to execute action \( a' \)).

### 3 A distributed approach to plan execution

The execution of the MAP is a critical task as the agents perform actions concurrently and they need to coordinate their activities in order not to violate the constraints defined during the planning phase.

Moreover, in a distributed system is quite natural not only to have a distributed execution of the plan, but also a distributed supervision of the plan execution by requesting that each agent performs a local control loop on the progress of the actions it has to execute (see for example, [Wittenveen et al., 2005; Micalizzi and Torasso, 2007a]).

We adopt a similar approach where a MAP \( P \) (which globally achieves a complex goal \( G \)) is decomposed in a number of sub-plans \( P^i \), each of which is assigned to the agent \( i \).

**Local Plans.** The decomposition can be easily done by selecting from \( P \) all the actions an agent \( i \) has to execute. Formally, the sub-plan for agent \( i \) is the tuple \( P^i=\langle A^i, E^i, CL^i, CC^i, NC^i \rangle \) where \( A^i, E^i \) and \( CL^i \) have the same meaning of the sets \( A, E \) and \( CL \), respectively, restricted to actions assigned to agent \( i \). In particular \( A^i \) includes two special actions \( a_0^i \) and \( a_\infty^i \) which specify, respectively, the initial and final conditions for the sub-plan \( P^i \). \( T^i \) is a set of incoming (outgoing) causal links \( a \rightarrow a' \) where \( a' \) belongs to \( A^i \) and \( a(a') \) is assigned to another agent \( j \) in the team.

We consider the time as 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 assumption is also made in [Wittenveen et al., 2005; Micalizzi and Torasso, 2007b]). At a given time \( t \), an agent \( i \) can execute just one action \( a \) (in the following the notation \( a^i \) will denote the action executed by agent \( i \) at time \( t \)). After the execution of action \( a^i \) the agent \( i \) may receive a set of observations, denoted as \( obs_{i,t+1} \), relevant for the status of \( i \) itself.

We assume that an agent \( i \) executes its next action \( a \) as soon as the preconditions of \( a \) are satisfied: the notion of action preceditions will be formalized in the following section.

**Coordination during plan execution.** As said above, the agents need to coordinate during the plan execution to avoid the violations of constraints defined in the MAP. The coordination can be achieved in an efficient way by exploiting the causal links defined in the MAP and maintained by each agent in the definition of its own sub-plan.

Intuitively, coordination is required when an agent \( i \) provides another agent \( j \) with a service \( q \); technically, this case is encoded by a causal link \( l : a \xrightarrow{\text{res}} a' \) in the MAP \( P \) (where \( a \in A^i \) and \( a' \in A^j \)); note that, after the MAP decomposition, \( l \) belongs both to \( T^j_{a'_{\infty}} \) and to \( T^i_{a_{\infty}} \). As a consequence of the partial observability, an agent can observe just the direct effects of the action it executes; therefore, after the execution of \( a \), the agent \( i \) must be able to observe the achievement (or the absence) of service \( q \) and must notify agent \( j \) whether the service has been provided or not. Also the consistent access to the resources is a form of coordination which, as mentioned above, is ruled by causal links labeled with the service \( free(res) \) (where \( res \) is a resource). This means that, after the execution of \( a \), the agent \( i \) releases the resource \( res \) and informs \( j \) about the actual status of \( res \). The agent \( j \) becomes the owner of the resource \( res \); i.e., only \( j \) can access \( res \).

Note that, since the system is distributed, an agent does not have a global view of the status of all system resources, but it knows just the status of the resources it holds. Thus, after the releasing of the resource \( res \), the agent \( i \) will not have access to the actual status of \( res \). In the following we will denote as available resources \( A\forall res(i,t) \) the subset of resources assigned to agent \( i \) at time \( t \); i.e., only agent \( i \) observes and knows the actual status of those resources. Obviously, the set of available resources depends on time \( t \) as this set changes according to the actions executed so far by the agent \( i \).

Alban Graastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Figure 1: The MAP $P$ to be supervised.

Observe that cooperation is required not only in nominal conditions (when a service has been provided), but also in anomalous situations (when services are unavailable) to prevent that agents wait indefinitely for services which will never be provided; in section 5 this issue will be discussed and a strategy of failure propagation will be presented.

**Running Example.** Along the paper we use a simple example from the blocks world for illustrating the basic concepts of the proposal. Let us consider a team of three agents $A_1$, $A_2$ and $A_3$, which cooperate to move the block $B_1$, $B_2$, $B_3$ and $B_4$ from a source position $S$ to the target location $T$. The global goal requires that $B_1$, $B_2$ and $B_4$ are put down in position $T$, and $B_3$ is put on the top of block $B_2$ as shown in Figure 1.

Source and target positions are critical resources as only one agent at a time can load/unload a block within them. We assume that the plan $P$ to be monitored satisfies all the constraints on the use of the resources. Moreover, we assume that an agent can handle at most one block at each time.

Figure 1 shows a possible instance of a MAP which achieves the target configuration of blocks. All the agents are initially located in a parking area $K$. The agent $A_1$ is responsible for moving the block $B_1$ (see the load and unload actions $2$ and $4$, respectively). The agent $A_2$ moves first block $B_2$ and subsequently the block $B_4$. Finally, the agent $A_3$ puts block $B_3$ on the top of block $B_2$.

The plan is a DAG whose nodes correspond to actions; edges can be precedence links or causal links: for the sake of readability, precedence links are dashed arrows; internal causal links (i.e. links in $CL'$) are dashed-double-dotted arrows; finally, external causal links are solid arrows. Causal links are labeled with the services an action provides to another one: e.g. the causal link from action 2 to action 4 is labeled with the service $\text{move}(A_1, B_1)$, which is both one of the effects of action 2 and one of the preconditions for the execution of action 4. Finally, the dotted rectangles specify which actions are included in the sub-plans assigned to the three agents.

From the plan in Figure 1 it is easy to see that $\text{AvRes}(A_2, I) = \{S, T\}$; i.e., after the execution of the pseudo-action $a_0$, the resources $S$ and $T$ are assigned to agent $A_2$. An agent is not required to immediately use all the resources it holds while it is carrying on an action; but it can acquire resources which it will use in the future. The distribution of the resources among the agents imposes also an order in the execution of the actions. For example, the action 2 will be executed as soon as its preconditions will be satisfied, this implies that the action 8 must be already executed, in fact the action 8 provides action 2 with the service $\text{free}(S)$.

4 Monitoring the execution of a MAP

Each agent of the team monitors the execution of the actions it is responsible for in order to estimate its own status after the execution of every action and to detect their outcome. The formalization of the monitoring process requires first the introduction of some important concepts.

**Agent status.** Intuitively, the system status can be expressed in terms of the status variables of the agents in the team $T$ and of the status of the system resources $RES$. However, the distributed approach to the supervision prevents the adoption of a global notion of status while allows a local view based on a single agent.

The status of agent $i$ is expressed in terms of a set of status variables $VAR'_i$, which is partitioned into three subsets $END'_i$, $ENV'_i$ and $HLT'_i$. $END'_i$ and $ENV'_i$ denote the set of endogenous (e.g., the agent’s position) and environment (e.g., the resources status) status variables, respectively. Note that, because of the partitioning, each agent $i$ has to maintain a private copy of the resource status variables; more precisely, for each resource $res_k \in RES$ ($k \in 1, \ldots, |RES|)$ the private variable $\text{res}_{k,i}$ is included in the set $ENV'_i$. Of course, since a resource variable is duplicated in as many copies as the agents, maintaining the consistency among all these copies could be a challenging issue. In our approach, however, this issue does not arise as conflicts for accessing the resources are solved at planning level. In particular, the causal links are set in such a way that just an agent $i$ at a time $t$ can access a resource $res_k$. As a consequence, when resource $res_k$ belongs to $\text{AvRes}(i, t)$, the status of $res_k$ is known only by agent $i$; whereas for the other agents $j \in T \setminus \{i\}$ the status of the
The Role of Agent Diagnosis in Multi-Agent Plan Repair

resk is unknown. The consistency among the private copies is maintained since at each time only the private copy resk,i determines the status of resk.

Since we are interested in monitoring the plan execution even when action failures occur, we introduce a further set of variables in order to model the agent faults which may explain the action failures. HLTi denotes the set of variables concerning the health status of an agent functionalities; in particular, for each agent functionality f, a variable v_f ∈ HLTi represents the health status of f, the domain of variable v_f is the set {ok, abn1, . . . , abn_n} where ok denotes the nominal mode while abn1, . . . , abn_n denote non nominal modes.

It is worth noting that the observations obsi convey information about just a subset of variables in VARi. First of all, an agent can directly observe just the status of its available resources. Moreover, the observations obsi provide in general the value of just a subset of variables in ENDi, whereas the variables in HLTi are not directly observable and their actual value can just be inferred. Given this partial observability, at each time t the agent i can just estimate a set of alternative states which are consistent with the received observations obsi; in literature this set is known as belief state and will be denoted as Bi.

Action models. As discussed in [Jensen and Veloso, 2000], the model ∆(ai) of an action ai (to be executed by agent i at time t) is characterized by three parts: a set var(ai) ⊆ VARi of state variables, a set pre(ai) of preconditions and a set eff(ai) of effects, where both preconditions and effects are constraints defined over the set var(ai). Since an action may fail, the action model ∆(ai) must model both the nominal and the anomalous effects. In particular, the anomalous effects of an action are in general non deterministic (i.e., a failure may affect the action execution in different ways, possibly not directly observable); therefore the action model ∆(ai) can be seen as a transition relation where every tuple d ∈ ∆(ai) models a possible change in the status of agent i, which may occur while i is executing ai. Each tuple d has the form d = (si, s+1). In particular, si and s+1 represent two agent states at time t and t+1 respectively; each state is a complete assignment of values to the state variables in var(ai).

Given the action ai, healthVar(ai) = HLTi ∩ var(ai) denotes the set of variables representing the health status of the functionalities which directly affect the outcome of action ai. The healthy formula healthy(ai) of action ai is computed by restricting each variable v ∈ healthVar(ai) to the nominal behavior mode ok and represents the nominal health status of agent i required to successfully complete the action itself.

Definition 1 Given action ai, the set of its nominal effects is nominalEff(ai) = {q ∈ eff(ai)| pre(ai) ∪ healthy(ai) ⊆ q}.

On the contrary, when at least one variable v ∈ healthVar(ai) assumes an anomalous mode (i.e., a functionality is not in the nominal mode), the behavior of the action may be non deterministic and some of the expected effects may be missing. Note that the non deterministic behavior of the actions is easily captured by the relational formalism; in fact ∆(ai) models both the nominal and the faulty behavior of action ai.

The state estimation process. The estimation process aims at predicting the status of an agent after the execution of an action ai; because of non determinism in the action model and partial observability the estimation process in general can just provide a set of states of the agent rather than a single state. The estimation can be formalized in terms of the Relational Algebra operators as follows.

Definition 2 Given agent i, let Bi be the belief state of agent i, let ∆(ai) the model of the action the agent has to execute at time t, the agent belief state at time t + 1 results from:

Bi+1 = projectionAYERi selectionobsi (Bi JOIN ∆(ai))

The join operation Bi JOIN ∆(ai) represents the prediction step as it has the effect of estimating the set of all the possible agent states at time t + 1. This set of predictions is refined by the selection selectionobsi, which prunes off all those estimates which are inconsistent with the agent observations. Finally, the belief state at time t + 1 is obtained by projecting the resulting estimates over the status variables of agent i at time t + 1.

Action outcome. Given the belief Bi+1, the agent i has to determine the outcome of the action ai. The outcome of an action is either succeeded or failed; in particular, action ai is considered succeeded when all its nominal effects have been achieved after its execution; more formally:

Definition 3 The outcome of action ai, is succeeded iff ∀q ∈ nominalEff(ai), ∀s ∈ Bi+1, s = q.

In order to be conservative, we consider action ai successfully completed only when all the atoms q in nominalEff(ai) are satisfied in every state s in Bi+1, i.e., when all the nominal effects of ai hold in every possible state estimated after the execution of the action. Of course, when we cannot assert that action ai is succeeded we assume that the action is failed.

Of course, the estimation process and the subsequent outcome evaluation are repeated at each time instant till the agent i reaches its goal or an action failure is detected. An in-depth description of the monitoring process is reported in [Micalizio and Torasso, 2007a].

5 Diagnosing and Revising a MAP

Plan Analysis: Motivations. Since the framework we propose aims at maintaining alive the system despite the occurrence of action failures, it is natural that the detection of a not nominal outcome should activate a plan analysis in order to: 1) explain the failure in terms of faults in some agent functionalities (agent diagnosis); 2) estimate the impact of the failure in the plan (failure propagation) and 3) repair (if possible) the plan in order to limit the impact of the failure (plan repair).

In fact, after the failure of action ai, agent i has to take into account that its health status is not nominal; hence the actions in P[i,a1, . . . , ai−1] (i.e., the next actions of agent i) may have non deterministic effects and may lead the system in dangerous or undesirable conditions. For example, assume that the action 2 fails as a consequence of a fault in the handling functionality of agent A1 and let’s assume that this failure is properly diagnosed so that handling=broken is the only possible explanation. In principle, a failure in the handling functionality does not affect the execution of the subsequent action.
3; in fact, healthVar(move(A1, S, T)) does not include the handling variable, so the healthy formula for action 3 is satisfied. Therefore, the move action is successfully executed by the agent A1; however, after this step, the agent cannot execute the next action 4 as one of its preconditions (the atom loaded(A1, B1)) is missing. Observe that the agent A1 is in a deadlock condition as it will wait for a service that only the agent itself can provide; moreover, the action 4 is not failed, thus no diagnostic or recovery mechanisms are activated. At the same time, the resource T is locked and no other agent can access it: the local failure of action 2 may cause a global plan failure.

This simple example shows that all the properties and guarantees concerning the use of resources and the cooperation among agents in the original plan P can be lost after the occurrence of an action failure. To re-establish these properties, as soon as the failure of action A2 is detected, the sub-plan P′ needs to be revised in order to overcome the harmful effects of the failure (through a plan recovery strategy) or at least to limit these effects by leading the system in a safe status, where the agent i releases all the resources in the set ArRes(i, t). While a discussion on the possible recovery strategies is out of the scope of this article, in the following of this section we will discuss the role of the agent diagnosis in bringing the agent into a safe status.

**Agent Diagnosis.** Once the agent belief state B′_{i+1} has been estimated, the agent i can determine the outcome of action A2 according to Definition 3. In case the outcome of A2 is assumed to be failed, a diagnostic process is activated in order to infer a set of possible explanations for such a failure. The diagnostic process singles out which fault (or a combination of them) in agent i may be the cause of the failure of action A2. In particular, the agent diagnosis explains the failure of action A2 in terms of the status variables in healthVar(α). In the relational framework we propose, the agent diagnosis D′_{i+1} can be inferred from the belief state B′_{i+1} as follows:

**Definition 4** Given the action A2, let B′_{i+1} be the belief state of agent i at time t + 1, the agent diagnosis D′_{i+1} is projection_{healthVar(α)}(B′_{i+1}).

Each tuple d ∈ D′_{i+1} is an assignment of values to the variables in healthVar(α), moreover, according to Definition 2, every assignment d is consistent with the observations obs′_{i+1}; hence, every tuple d is a possible explanation for the failure of action A2.

Since the system is only partially observable and the health status variables cannot be directly observed, the agent diagnosis is in general ambiguous (i.e., it involves a number of alternative explanations).

**Missing Goals.** As said above, the execution of the actions in the plan segment P*[a_{i+1}, a_{∞}] may lead the system in dangerous conditions; however, the other agents may be affected even when these actions are not executed. In fact, there may exist outgoing links, starting from an action in P*[a_{i+1}, a_{∞}] which refer to services that agent i should provide to other agents in the team but that it can no longer provide as a consequence of the failure of action A2. The set of these services is denoted as the set of missing goals; singling out these services is essential as, in principle, it would be sufficient to find an alternative way to provide them in order to reach the MAP’s global goal G despite the occurrence of the action failure. To formally characterize the notion of missing goals we introduce the concept of primary effect.

**Definition 5** Given an action a ∈ A, the effect q ∈ eff(a) is said primary if

i. q ∈ nominalEff(a) and

ii. it holds either

- q ∈ pre(a∞) or

- 3 a causal link l ∈ CL such that l : a → a′ where a and a′ are actions assigned to different agents.

Namely, the effect q of action a is said primary when q is a nominal effect of a (condition i), and when either q is a an atom which appears in the global goal G (i.e., q is a precondition of the special action a∞) or q is a service that an agent provides to another one. In general, given an action a, primary(a) denotes the (possibly empty) set of primary effects provided by a.

The set of missing goals can be determined as the set of primary effects which should be provided by the actions in the segment P*[a_{i+1}, a_{∞}].

**Definition 6** Let RP*[a_{i+1}, a_{∞}] the set missingGoals(i) of goals that agent i can no longer achieve is: missingGoals(i) = \bigcup_{a′ \in RP} primary(a)

**Failure Propagation.** The failure propagation aims at singling out all those actions threatened by the absence of the services in the set missingGoals; i.e., actions which are threatened by causal dependencies. Intuitively, an action a is threatened through a causal links l : a′ → a when it is no longer guaranteed that the action a′ provides the service q; this may happen either because a′ is failed or because a′ is in turn threatened. More formally,

**Definition 7** Given the failed action a′, the set cThreatenedActs(a′) of actions threatened through causal links is

\[ cThreatenedActs(a′) = \{ a ∈ A | \]

i. a′ ≺ a and

ii. \exists a causal link l : a′ → a, l ∈ CL, such that

- q belongs to missingGoals(i) or

- a′ ∈ cThreatenedActs(a′) \}

Observe that, in the above definition, the global set of causal links CL is involved; thereby the agents in the team need to communicate to compute the set of actions cThreatenedActs(a′). However, the communication does not represent a significant source of overhead as it is ruled by the causal links themselves; thus it is easy to prove the following property:

**Property 1** Given the failure of action a′, the process for propagating this failure in global plan P is polynomial in the number of causal links in CL.

**Reaching the Safe Status.** So far we have discussed the steps of agent diagnosis and plan failure in the very unlucky condition when agent i is unable to execute any action after the occurrence of a failure. However, in general, an agent is
still able to do something even if its health status is not nominal. Exploiting this possibility, the agent $i$ could try to reduce the impact of its failure (i.e., the result of the failure propagation) by moving into a safe status where it does not represent a latent menace for the other agents: the activation of the failure propagation from a safe status will affect, in general, a smaller number of actions.

Intuitively, given the failure of action $a_i^l$, a safe status $S^l$ for $i$ is a state where all the resources and objects currently acquired by $i$ are released and made available to other agents; that is, all the resources in the set $AvRes(i, t)$. To reach $S^l$ the agent $i$ has to revise $P^i$, i.e., a re-planning step is required; in particular, this re-planning step has to take into account the assumed health status of agent $i$. For this reason, we adopt the conformant planner (see e.g., [Cimatti and Roveri, 2000]) presented in [Micalizio and Torasso, 2007b]: by relying on the (usually ambiguous) agent diagnosis, the planner synthesizes a plan whose actions can be successfully executed by the agent $i$ no matter the actual health status of the agent is. The choice of a conformant planner is needed as it guarantees that, when a plan recovery $P^{i*}$ exists, this plan can be successfully executed by the agent $i$ (under the hypothesis that no other fault occurs during the recovery actions).

Due to space reasons, we cannot provide details about the conformant planning process. We just want to remark that the pieces of information exploited by the conformant planner are the belief state $B^i_{t+1}$, which represents the (usually ambiguous) initial state of the planning process; the safe status $S^l$, which represents the goal to be achieved; and the set of conformant actions which the agent $i$ is still able to execute by taking into consideration the agent diagnosis $D^i_{t+1}$. The output of the planner is the plan $P^{i*}$ if a conformant solution exists, an empty plan otherwise. 

Revising the interrupted sub-plan. When a conformant solution does not exist the agent is the worst condition as described above. Conversely, when a conformant plan repair $P^{i*}$ exists, the agent $i$ executes this plan in lieu of the segment $P^i[a^l_{i+1}, a^l_{\infty}]$: however, replacing the old plan with the new one, the agent $i$ has to inform the other agents

1) for every incoming link $l_{get} : a^l_{i} \xrightarrow{free(res)} a^l_{get}$ (acquisition of resource res) finds the corresponding outgoing link $l_{rel} : a^l_{rel} \xrightarrow{free(res)} a^l_{rel}$ (relinquishment of res). Of course it must hold that $a^l_{i} \prec a^l_{rel} \prec a^l_{get} \prec a^l_{rel} \prec a^l_{k}$; moreover, between the two actions $a^l_{get}$ and $a^l_{rel}$ there not exist any other actions whose effect is $free(res)$.

2) substitute $l_{get}$ with the link $l' : a^l_{free(res)} \xrightarrow{\rightarrow} a^l_{k}$; i.e., notify to agent $j$ that the resource $res$ is made available to agent $k$.

3) substitute $l_{rel}$ with the link $l'' : a^l_{free(res)} \xrightarrow{\rightarrow} a^l_{k}$; i.e., notify the agent $k$ that the resource $res$ will be made available by agent $j$.

that it will not acquire any other resource in the future. In fact, the plan $P^{i*}$ leads the agent $i$ in a state where all the resources in $AvRes(i, t)$ are released; however, according to the original actions in $P^i[a^l_{i+1}, a^l_{\infty}]$, the agent may acquire other resources necessary for carrying on the actions that have to be disregarded and substituted with the plan $P^{i*}$. Consider an incoming link $l_{get} \in T_{in}^i$ such that $l_{get} : a^l_{i} \xrightarrow{free(res)} a^l_{get}$ where $a^l_{rel} \in P^i[a^l_{i+1}, a^l_{\infty}]$; as said above this link represents the fact that agent $i$ will acquire the resource $res$ after the execution of action $a^l_{i}$. Of course, in the original plan $P^i[a^l_{i+1}, a^l_{\infty}]$ there must exist an analogous link $l_{rel} \in T_{out}^i$ $(l_{rel} : a^l_{rel} \xrightarrow{free(res)} a^l_{k})$ through which the agent $i$ releases the resource $res$ to another agent $k$: where possibly $j\neq k$. However, in the plan $P^{i*}$ the link $l_{rel}$ is missing hence the agent $i$ will acquire a resource which will be never relinquished. Therefore, it is essential not only that agent $i$ releases all the resources in $AvRes(i, t)$, but that the agent $i$ does not acquire any further resource in the future.

To this end, the portion of plan $P^i[a^l_{i+1}, a^l_{\infty}]$ is revisited as in Figure 2. It is worth noting that, albeit the agent $i$ revises its own plan $P^i$, such process impacts also the local plans of other agents in the team. In fact, when the agent $i$ notifies other agents that a link must be revised, it requires the revi-
sition of the plan of these agents too (see steps 2 and 3). Since we consider just cooperative agent teams, we assume that an agent is always willing to revise its own subplan according to the requests of another agent in the team.

Running Example. To make clear the processes of agent diagnosis and plan revision, let’s consider again the previous example from the blocks world and assume that the move action 3 (assigned to agent A1) fails. Observe that if the agent A1 did not properly handle this failure, only the block B2 would be positioned in the target location T; in fact, since agent A1 occupies the resource S, no other agent can load the blocks to be still moved. In this example, we show how the agent diagnosis is exploited to revise the local plan of agent A1 and to limit the harmful effects of the failure. First of all, the agent A1 can detect the failure of action 3 as it receives the observation position=S; i.e., the agent has not moved. Once detected the failure, the agent diagnosis is inferred by projecting the current belief of A1 over its status variables; in this example the (preferred) diagnosis for the failure of action 3 is \( D^{A1} = \{\text{mobility = slowdown} \lor \text{power = reduced} \} \). In fact, we assume that a move action can fail not only as a consequence of blocking faults (e.g., \text{mobility = broken or power = flat}), but also as a consequence of degraded conditions, under which an agent can move iff it is unloaded. In the case under consideration, it is easy to see that the agent diagnosis \( D^{A1} \) is inferred by taking into consideration that agent A1 is loaded with block B1 during the execution of the move action 3. Observe that the agent diagnosis \( D^{A1} \) partially impacts also the load/unload actions; in fact we assume, that an agent can unload a block even when its power functionality is degraded, but it cannot load a block. Of course, load and unload actions fail when power is flat; while the mobility functionality has not any role in these action types. After the inference of the agent diagnosis \( D^{A1} \), the agent A1 has to release all the resources and blocks it is currently holding. In this case, the safe status agent A1 tries to reach is \( S^{A1} = \{\text{AT (B1, S)}, \text{FREE (S)}\} \). In order to reach the safe status, agent A1 invokes the conformant planner which, by taking into account the agent diagnosis \( D^{A1} \), infers the plan to safe status \( P^{A1} \) (see Figure 3). The plan to safe status \( P^{A1} \) consists of the two actions R1 and R2; the first has the effect of positioning the block B1 in the location S; the latter has the effect of releasing the resource S, which becomes available for agent A3.

Figure 3 shows also the result of the causal link revision which the agent A1 performs over the original local plan. In fact, since the agent A1 will no longer get the resource T, the links between actions 10 and 4 and between actions 5 and 17, showed in Figure 1, are substituted with the direct link between actions 10 and 17 in Figure 3. Finally, it is easy to see that the set of missing goals consists of the atom AT (B1, T); therefore, a recovery strategy should be focused on finding an alternative way to reach this atom by exploiting the agents in the team, possibly discarding the agent A1 whose degraded health status may prevent the execution of some types of actions.

Let’s consider another case of execution of the same plan \( P \): let us assume that the move action 8 (assigned to agent A2) fails similarly to the action 3 of the previous example. This means that there exists a plan to a safe status \( S^{A2} \), which is similar to the previous one; moreover the agent A2 will not acquire the resource T in the future. It is worth noting that, albeit the failure of action 8 is handled similarly to the failure of action 3, the failure of action 8 has a wider impact on the plan. In fact, the set of missing goals consists of the atoms: \( \{\text{AT (B2, T)}, \text{AT (B4, T)}\} \); namely, the agent A2 is unable to move blocks B2 and B4 in the target location T. The propagation of the missing goals in the plan (e.g., through the causal link between actions 9 and 17) makes evident that the failure of 8 impacts on the actions of agent A3, which will be unable to put block B3 on the top of block B2. In this case, only the agent A1 is able to complete its local plan moving B1 from S to T.

6 Implementation and preliminary results

We have implemented the proposed methodology by exploiting the symbolic formalism of the Ordered Binary Decision Diagrams (OBDDs), in order to encode the relations representing the belief state of the agents and the non deterministic models of the actions. As discussed in [Micalizio and Torasso, 2007a], the processes of monitoring and diagnosis are implemented in terms of standard OBDD operators. The OBDDs play also a critical role in the synthesis of the conformant plan; in fact the OBDDs are exploited for efficiently encoding a complex structure, which maintains all the partial conformant plans built at each intermediate step of the search process (see [Micalizio and Torasso, 2007b] for an in-depth discussion).

In order to prove the effectiveness of the proposed methodology, we have simulated a service-robot scenario where a team of robotic agents offer a “mail delivery service” in an office-like environment. Resources are parcels (which can be light or heavy), clerks’ desks, doors, and one or more repositories. Resources are constrained: desks, doors and repositories can be accessed only by one agent per time; moreover, at most one parcel can be put on a desk. An agent can move two light parcels or one heavy parcel from a repository to a desk or from a desk to a repository or to another desk.

In our experiments we have considered a fairly large environment as it involves 30 critical resources. We have devised 10 plans involving up to 6 robotic agents, where each plan has in average, 60 actions to be executed and involves more than 100 causal links. In each plan we have simulated the occurrence of up to 3 agent faults, even simultaneous.

The diagnostic system has been implemented in Java JDK 1.6 and exploits the JavaBDD package for symbolically encoding and manipulating OBDDs. The robotic agents are simulated in a software environment and are implemented as threads running on the same PC.

The preliminary results collected so far are very promising. First of all, the solution is very efficient: the monitoring step in nominal conditions requires on average 26 msec. per time instant; whereas in case of action failure the supervision

---

In Figure 3 the actions highlighted with a gray background are those already executed at the time of the failure of action 3.

---

1In Figure 3 the actions highlighted with a gray background are those already executed at the time of the failure of action 3.

2http://sourceforge.net/projects/javabdd

3Intel Pentium 1.86 GHz, RAM 1 GB, Windows XP OS.
process, which includes diagnosis, plan revision and failure propagation, requires up to 100 msec. In the worst cases; this result provides an experimental confirmation of Property 1, which relates the computational cost of the propagation to the number of causal links in the plan. Concerning the performance of the conformant planner, we have limited to 5 actions the maximum length of a plan to safe status. The conformant planner requires on average 897msec. (and up to 2141 msec. in worst case), to find a conformant solution; whereas, when a solution does not exist, the conformant plan employs only 266 msec. on average to determine that no conformant actions are applicable.

To show the advantage of exploiting the agent diagnosis to limit the harmful effects of a failure, we have compared the execution of the 10 cases when the revision policy is set to on and when it is set to off. In the off condition the percentage of actions actually executed is 20% of the initial global plan \( P \) while 37% is the percentage of the sub-goals in \( G \) which have been achieved. Under on condition the percentage of executed actions grows to 70% and, as a consequence, the percentage of sub-goals actually achieved is 76%.

7 Discussion and conclusions.

In this paper an approach for monitoring and diagnosing the execution of a multi-agent plan has been discussed and formalized. Similarly to other approaches ([Wittenveen et al., 2005; Micalizio and Torasso, 2007a]), the proposed solution adopts a distributed architecture where each agent is responsible for monitoring and diagnosing the actions it executes.

The distribution allows to extend some of the model-based methods developed for distributed component-based systems (e.g. [Pencolé and Cordier, 2005]) to the multi-agent scenario. In particular, the paper introduces the notion of agent diagnosis for explaining the failure of an action in terms of faults in the functionalities of the agent responsible for that specific action.

While other works ([Wittenveen et al., 2005; Kalech and Kaminka, 2007; Micalizio and Torasso, 2007a]) are mainly focused on the task of diagnosis, this paper discusses also a plan revision process, which (relaying on the notion of agent diagnosis) is able to limit the harmful effects of an action failure leading the system in a safe status, where more appropriate recovery strategies can be activated. Moreover, the approach is able to infer the set of missing goals and the set of threatened actions, which play an essential role for any strategy conceived for recovering the execution of the multi-agent plan, since they focus the process of searching alternative plans.

In this paper we have assumed that an agent receives, at each time instant, a sufficient amount of observations for determining the outcome of the last action it has executed. However, the framework can be extended by relaxing this assumption as discussed in [Micalizio and Torasso, 2008]; as a consequence the agent diagnosis task becomes more challenging as an agent cannot consider just a belief state but a trajectory consisting of a set of past belief. In this framework, the outcome of an action can be in a pending status as long as the observations a later times allow to infer whether previous actions have achieved or not the expected effects.

A final comment concerns the need of communication among the agents for performing the monitoring and the diagnosis. As pointed out in [Kalech and Kaminka, 2007], distributed solutions may suffer from high computational costs due to the communication overhead. This does not happen in our approach because the local plans maintain trace of the causal links, which precisely define what message has to be sent and to what agent.

The preliminary experimental results show that the proposed methodology is adequate to promptly react to an action failure and to actually mitigate the harmful effects of the failure. These results pave the way for testing the approach in other relevant domains such as the air traffic control domain and the space exploration scenario.

References


Using Bayesian networks for Candidate Generation in Consistency-based Diagnosis

Sriram Narasimhan & Ole Mengshoel
UC Santa Cruz & USRA RIACS
M/S 269-3, NASA Ames Research Center,
Moffett Field, CA 94043
Sriram.narasimhan-1,ole.j.mengshoel@nasa.gov

Abstract
Consistency-based diagnosis relies on the computation of discrepancies between model predictions and sensor observations. The traditional assumption that these discrepancies can be detected accurately (by means of thresholding for example) is in many cases reasonable and leads to strong performance. However, in situations of substantial uncertainty (due, for example, to sensor noise or model abstraction), more robust schemes need to be designed to make a binary decision on whether predictions are consistent with observations or not. However, if an accurate binary decision is not made, there are risks of occurrence of false alarms and missed alarms. Moreover when multiple sensors (with differing sensing properties) are available the degree of match between predictions and observations of each sensor can be used to guide the search for fault candidates (selecting candidates “closer” to sensor observations that are more likely to be inconsistent with corresponding predictions).

Using Bayesian networks, we present in this paper a novel approach to candidate generation in consistency-based diagnosis. In our formulation, automatically generated Bayesian networks are used to encode a probabilistic measure of fit between predictions and observations. A Bayesian network inference algorithm is used to compute most probable fault candidates taking into account the degree of fit between predictions and observations for each individual sensor.

1 Introduction
Consistency-based diagnosis techniques [Hamscher, et al., 1992] compare model predictions against sensor observations in order to isolate faults. Structural and behavioral models are used to predict system behavior under hypothesized nominal and faulty conditions. The hypotheses whose predictions best match the sensor observations are reported as the diagnosis. Rather than looking at a pre-enumerated set of hypotheses, these approaches use techniques like conflict directed search and backtracking to maintain a short list of consistent hypotheses. This has been applied to discrete domains [De Kleer and Williams, 1987; Williams and Nayak, 1996; Kurien and Nayak, 2000], continuous [Gertler, 1988; Mosterman and Biswas, 1999] and hybrid [Narasimhan and Biswas, 2003] domains.

One of the assumptions in these approaches is that we can say with absolute certainty if predictions and observations are consistent with each other. For example, the Livingstone 2 system [Kurien and Nayak, 2000] uses monitors to make this decision and the TRANSCEND system [Mosterman and Biswas, 1999] uses a symbol generation scheme to make a decision that is statistically robust. If any inconsistencies are detected, information about the inconsistencies (including variables involved in the inconsistencies and possibly the direction or magnitude of the inconsistencies) is used to guide the search for alternate fault candidates. Fault candidates that resolve the observed discrepancies (resolving conflicts entailed by the discrepancies for example) are generated according to some user-defined criteria (typically based on prior probabilities and the size of the set of chosen candidates).

However, in a lot of situations it may be very difficult to determine if model predictions exactly match sensor observations. Some reasons for this difficulty include (i) imperfect/abstract models resulting in imprecise predictions, (ii) sensor noise resulting in imprecise observations, and (iii) uncertain operating conditions and environment resulting in imprecision predictions and observations. An error in making this binary decision (either reporting a discrepancy when there is none or not reporting a discrepancy when one exists) will result in erroneous diagnosis results. Additionally the actual degree of fit between predictions and observations for individual variables might provide useful diagnostic information allowing us to limit the search for fault candidates (resulting in faster diagnosis).

Some probabilistic approaches address this problem by setting up a Bayesian formulation (as opposed to the consistency-based approach) to solve problem [Dearden and Clancy, 2002; Hofbaur and Williams, 2004:]. Typically candidates have weights/probabilities associated with them and these weights are updated at each time step based on the model and observed values. Candidates that represent faults may be introduced in several ways including importance sampling [Dearden and Clancy, 2002] and using a consistency-based diagnosis scheme [Narasimhan, et al., 2004].
In this paper we propose an alternate approach using Bayesian networks, which attempts to solve this problem within a consistency-based framework but using a Bayesian network as a component. In our approach, a consistency-based diagnosis engine is the driving force. The engine is responsible for maintaining a set of candidates “consistent” with the observations seen so far. At each time step the engine tests each candidate for consistency with the current observations. Rather than looking for a binary decision on the consistency of the candidate, the consistency test is used to provide probabilistic measure of the degree of fit between predictions and observations for each observed variable. The engine then utilizes a Bayesian network (BN) that encodes the structure associated with the current model as well as probabilistic information in the form of prior probabilities of faults and the probability of fit for observable variables (computed in the consistent testing step earlier). The BN can then be queried for the most probable assignment of values to all variables, a subset of which correspond to faults in the system. This information can then be used to update the candidate set maintained by the diagnosis engine.

In this paper we will focus on a specific consistency-based diagnosis system called HyDE [Narasimhan and Brownston, 2007] and show our approach works in that framework. However the ideas are general and can be adapted to other consistency-based diagnosis systems. We will describe the makeup of the BN, how it can be constructed automatically from existing models in the HyDE framework (this is the only part that would be different for a different diagnosis technology) and how it can be integrated with a consistency-based diagnosis engine. Initial experiments, with a two tank system, show a significant improvement in diagnostic accuracy when our novel approach is used.

The rest of the paper is divided as follows. Section 2 presents some background on BN and the consistency-based diagnosis paradigm we will be assuming. Section 3 presents the Hybrid Diagnosis Engine (HyDE) and its diagnosis architecture. Section 4 presents our novel approach of using BNs for candidate generation. Section 5 presents some examples and results from using this combined approach. Section 6 presents conclusions and ideas for future work.

2 Background

2.1 Consistency-based Diagnosis

Several interpretations of consistency-based diagnosis exist in the literature [Hamscher, et al., 1992]. In order to take into account the hybrid and dynamic nature of the systems being diagnosed, we will be using the following representation of consistency-based diagnosis as our basis. We assume that the consistency-based diagnosis uses a “generate and test” paradigm to detect and isolate faults. The diagnosis engine maintains a set of consistent candidates which is updated at each time step by adding or pruning candidates based on the observations from sensors. The candidates represent hypotheses about faults that have occurred in the system with associated time stamps. At each time step, candidates in the candidate set are tested for consistency against observations available at that time step. If a candidate is found to be inconsistent with observations, it is dropped from the candidate set. New candidates are generated by backtracking in the model from the point of inconsistency (typically an intermediate step of generating conflicts is used). The newly generated candidates can be tested and added to the candidate set if found to be consistent with observations. The test for consistency uses models that can be used to predict what the system is expected to do.

2.2 Bayesian networks

A Bayesian network (referred to as BN in the rest of this paper), or a belief network, is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. The term “Bayesian networks” was coined by Judea Pearl [Pearl, 1985] to emphasize three aspects:

1. The often subjective nature of the input information.
2. The reliance on Bayes’s conditioning as the basis for updating information.
3. The distinction between causal and evidential modes of reasoning, which underscores Thomas Bayes’s posthumous paper of 1763.

Bayesian networks are directed acyclic graphs whose nodes represent variables, and whose arcs encode conditional independencies between the variables. Nodes can represent any kind of variable, be it a measured parameter, a latent variable or a hypothesis. If there is an arc from variable $x_i$ to another variable $x_j$, $x_i$ is called a parent of $x_j$, and $x_j$ is a child of $x_i$. Associated with each variable $x_i$ is a joint probability distribution which specifies the probability of $x_i$ taking each value in its domain for all possible value assignments for the parents of $x_i$.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Because a BN is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the BN can be used to find out updated knowledge of the state of a subset of variables when other variables values (called evidence) are known. This process of computing the posterior distribution of variables given evidence is called probabilistic inference.

Formally, BN can be defined as $BN = ((X), (E), (P))$ where $X=\{x_1, x_2, \ldots, x_n\}$ are the $m$ variables in the BN with associated conditional probability distributions $P=\{p_1, p_2, \ldots, p_n\}$ and $E=\{e_1, e_2, \ldots, e_n\}$ represent the $n$ arcs between variables in $X$ with $e_i = x_i \rightarrow x_k$ for $i \neq k$.

3 Hybrid Diagnosis Engine (HyDE)

Hybrid Diagnosis Engine (HyDE) is a model-based reasoning engine for hybrid (discrete + continuous) diagnosis. HyDE is able to diagnose multiple discrete faults using consistency checking between prediction from hybrid models and sensor observations. We first describe the models used by HyDE in its reasoning.
3.1 HyDE Models

A HyDE model is made up of the following elements:
1. The set $C$ of the components of the system.
2. The set $L$ of operating modes of all components called Locations.
3. The set $TR$ of allowed transitions between the locations of the same component. Each transition $tr \in TR$ is of the form $l_{\text{from}} \rightarrow l_{\text{to}}$, where $l_{\text{from}} \in C$ & $l_{\text{to}} \in C$ for some $c \subseteq C$ and $g$ is a guard indicating the conditions under which the transition may be taken. An empty guard is used to encode a special kind of transition called unguarded transition. Unguarded transitions can be used represent, among other things, faults in the system.
4. The set $V$ of variables and the set $VD$ of domains associated with the variables, specifying the allowed values (data types) for the variables.
5. The propagation model PM specifies the behavior of the system within a time step as relations over variables. This includes:
   a. Global model $PM_g = R_g(V)$, where $R_g$ is the global set of relations constraining values of variables. These relations are valid at all times.
   b. Local models $PM_l = R_l(V)$ for each $l \in L$, where $R_l$ is the set of local relations constraining values of variables. These relations are applicable only when the system (corresponding component) is in location $l$.
6. The integration model IM specifies the evolution of values each variable across time steps. It specifies how state variable values at one time step can be computed from state variable values and derivative (of state) variable values at the previous time step.
7. The dependency model DM specifies how variables in the model are influenced by local relations.

3.1 HyDE Reasoning

The reasoning algorithm in HyDE (illustrated in Figure 1) essentially maintains a set of candidates $D$. The goal of HyDE is to find the candidates that best match the observations seen so far. 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 instance. It associates all components with their current locations and all variables with their current values. The hybrid state history tracks hybrid states at beginning and end of all time steps of the system behavior evolution. The transition history tracks transitions taken by all components at all time steps. In order to avoid monotonic growth in the histories, a user-defined parameter called history window is used to restrict the length of history saved.

At each time step $t_i$ in the reasoning process, HyDE tests each candidate for consistency with observations at $t_i$. This involves computing the hybrid state at the end of $t_i$ ($HS_{t_i}$) from the HS at the beginning of $t_i$ ($HS_0$) and the model entailed by the HS. A subset of variable values in $HS_{t_i}$ corresponds to predictions for observed variables. These predicted values are compared against corresponding observations. If they are found to be inconsistent then a candidate generator is created which finds unguarded transitions (one or more) that can possibly resolve the inconsistency. A transition can possibly resolve an inconsistency if the relations from the source location of the transition directly or indirectly influence the variable found to be inconsistent. Since multiple transitions may resolve an inconsistency and typically multiple inconsistencies (across time) may occur after a fault, a search process is needed to identify the most important transitions. Importance is judged based on criteria like maximum prior probability and minimum size. A candidate manager is responsible for pruning candidates that were found to be inconsistent and adding candidates by querying the candidate generators. The reasoning algorithm can be summarized as follows:

1. Initialize consistent candidate set with the empty candidate $D_c = \{d_{\text{empty}}\}$ where $d_{\text{empty}} = (HS_0, \{\})$ and $HS_0$ is the initial hybrid state of the system which is assumed to be known and the system is assumed to be in nominal state (no unguarded transitions have been taken).
2. Repeat at each time step $t_i$ for each candidate $d_k \in D_c$
   2.1. Advance the hybrid state from the end of the previous time step ($HS_{t_i-1}$) to the beginning of the current time step ($HS_{t_i}$).
   2.2. Compute HS at end of time step ($HS_{t_i}$) from $HS_{t_i-1}$, current values of input variables ($U_i$), global constraints ($R_h$) and local constraints associated with current system location obtained from $HS_{t_i}$ ($R_{li}$).
   2.3. Compare sensor values for observed variables $V_{\text{obs}}$ with predicted values $V_{\text{obs}}(HS_{t_i})$ to identify inconsistent variables $V_{\text{inconsistent}} \subseteq V_{\text{obs}}$.
   2.4. In the dependency model, trace backwards from each $v \in V_{\text{inconsistent}}$ to identify all local relations that influence $v$. The locations associated with these relations together form a conflict (if the system is assumed to be in these locations then $v$ becomes inconsistent).

2.5. Possible transitions that can resolve a conflict are the unguarded transitions out of the locations that form the conflict. Compute a set of unguarded transitions $TR$ (selecting the best set based on user-specified ranking criteria) that resolve conflicts associated with all inconsistent variables $V_{\text{inconsistent}}$. $TR$ and the HS history from $d_k$ can then be
used to generate a new candidate \( d_{\text{potential}} \). \( D_{\text{potential}} \) can then be tested to see if it actually resolves the conflict by tracking its behavior prediction. If it is found consistent with the observations then it is added to the candidate set \( (D_c = D_c \cup \{d_{\text{new}}\}) \) else it is discarded.

2.6. Remove the inconsistent candidate from the candidate set \( (D_c = D_c - d_c) \).

For more details about the HyDE reasoning algorithm please refer to [Narasimhan and Brownston, 2007].

3.2 Bayesian networks for Candidate Generation in HyDE

When we look at the steps of the HyDE reasoning algorithm, step 2.3 assumes that we can make a binary decision on which observations are inconsistent with predictions. Based on this decision, new candidates may be added (and the inconsistent one eliminated) using a conflict directed search. This approach fails when inconsistencies cannot be detected accurately. It also fails to make use of the magnitude/degree of the inconsistency which might be a useful guide when searching for candidates. We propose a modified HyDE reasoning algorithm that uses the degree of inconsistency (rather than expecting to make a binary decision based on it) by constructing a BN and generating candidates to be tested by computing the most probable hypothesis in the BN.

The basic idea is to estimate a probabilistic measure of the (in)consistency between model predictions and sensor observations. Depending on the domain of the variable and sensor noise properties this may be achieved by simply thresholding or by other means like the probability distribution function of a Gaussian distribution (which we will see in the example later). In order to use this probabilistic measure effectively, we use an automatically constructed BN in place of the dependency model for candidate generation. The structure of the BN is determined by the propagation and integration model (Bullets 5 & 6 from the HyDE model description). The conditional probability distributions in the BN are obtained from prior probabilities of unguarded transitions and the estimated probabilistic measured of (in)consistency between predictions and observations.

We modify HyDE reasoning algorithm in the following ways. We add an initialization step (1a) that initializes the BN for the initial candidate.

1a. Initialize the BN\(_{\text{empty}}\) associated with nominal candidate with nodes for locations of components at start of time step 0. If \( l_i \) be the location of component \( c_i \) in HS\(_0\), then \( \text{BN}_{\text{empty}} = (\{x_{l_1}, \ldots, x_{l_n}\}, \{P_{c_1}, P_{c_2}, \ldots, P_{c_m}\}, \{\}) \) where \( P_{c_i} = \{p(c_i \in l_i) = 1.0, p(c_i \in l_k \neq i) = 0.0\} \)

Steps 2.3 through 2.6 are modified as follows:

2. Repeat at each time step \( t_i \) for each candidate \( d_c \in D_c \).

2.3. Compare sensor values for observed variables \( V_{\text{obs}} \) with predicted values \( V_{\text{hyp}}(\text{HS}_{t_i}) \) to assign probabilities to each variable being consistent. For \( v \in V_{\text{obs}} \) \( p(v=\text{CONSISTENT}) = \delta(v,v^*) \) where \( \delta \) is a user-customizable comparison function (for example thresholding). \( p(v=\text{INCONSISTENT}) = 1- p(v=\text{CONSISTENT}) \).

2.4. Construct the BN for \( t_i \) and for the transition from time step \( t_i-1 \) to \( t_i \) and append it to BN for the candidate \( D_{\text{new}} = D_{\text{new}} \cup \{(X_{t_i}, \{E_{t_i}, \{P_{t_i}\}) \text{ This step is discussed in more detail in the next section.} \}

2.5. Query BN\(_{\text{new}}\) for the Most Probable Explanation (MPE) which provides the most likely assignment of values for all variables in the BN. A subset of these variables corresponds to transitions taken by all components at all time steps in the time history window. This subset will be used to generate a new candidate \( d_{\text{new}} \) with hybrid state obtained from \( d_c \). If \( d_{\text{new}} = d_c \) then nothing needs to be done. However if \( d_{\text{new}} \neq d_c \) then \( d_{\text{new}} \) is added to the set of consistent candidate \( (D_c = D_c \cup d_{\text{new}}) \).

2.6. Check the probability of the \( d_c \) in the BN and if it falls below a certain threshold \( p_{\text{cutoff}} \) then remove \( d_c \) from \( D_c \) (\( D_c = D_c - d_c \)).

We describe the automatic construction of the BN models (step 3.4) for a specific candidate \( d_c \) at a specific time step \( t_i \) in section 4.

4 Automatic Generation of Bayesian networks

We noted that in step 3.4 of the modified HyDE reasoning algorithm presented in the previous section, the BN for each candidate \( d_c \) (BN\(_{\text{new}}\)) is augmented with the BN fragment at the current time step \( t_i \) (BN\(_{\text{new}}\)) and BN fragment for the transition from previous time step \( t_{i-1} \) to current time step \( t_i \) (BN\(_{\text{new}}\)\(_{t_{i-1} \rightarrow t_i}\)) BN\(_{\text{new}}\) = BN\(_{\text{new}}\) \( \cup \) BN\(_{\text{new}}\)\(_{t_{i-1} \rightarrow t_i}\).

We first describe the automatic generation of BN\(_{t_0} \) from the HyDE models and then describe the generation of BN\(_{t_{i-1} \rightarrow t_i}\).

4.1 Bayesian network within a time step

In order to generate the BN at a specific time step \( t_i \) for candidate \( d_c \), we find the system location predicted by \( d_c \) at the beginning of \( t_i \). This can be obtained from the hybrid state predicted by \( d_c \) at the beginning of \( t_i \) (HS\(_{t_i}\)). SL\(_{a} \) = \{I\(_{l_1}\), I\(_{l_2}\),...,I\(_{l_n}\} = \text{SL(HS}_{t_i}\)) where \( l_j \) corresponds to the location of component \( c_j \). The next step is to construct the constraint system model (RN\(_a\)) at \( t_i \) predicted by \( d_c \). This should already have been constructed in step 3.2 and can be re-used in the generation of BN\(_u\).

In the reminder of this section we will omit the subscript \( t_i \) for convenience. BN is computed as follows. The nodes \( \{X\} \) in the BN consist of nodes corresponding to variables in the model (\( X_v \)) and nodes corresponding to components in the model (\( X_{c} \)): \( X = X_v \cup X_c \)

- \( X_v = \{x_{v1}, x_{v2}, \ldots, x_{vn}\} \) where \( x_{vi} \in \{\text{CONSISTENT,INCONSISTENT}\} \) corresponds to variable \( v_i \) in the constraint system model.
\( X_c = \{x_{c1}, x_{c2}, \ldots, x_{cm}\} \) where \( x_{ci} \in \text{Locations}(l_i) \) corresponds to Component \( c_i \) in the constraint system model.

The arcs \([E]\) between nodes in \( X \) are computed based on the currently valid relations \( R = R_c \cup R_{SL} \). We use the following algorithm for generating arcs in the BN:

1. Create two variable lists, KNOWN (\( X_k \)) & UNKNOWN (\( X_u \)). Move all input and state variables to \( X_k \) and all other variables to \( X_u \).
2. Create a TOBEPROCESSED relations list (RT) and move all relations in \( R \) to RT.
3. Repeat until RT \( \neq \{\} \):
   a. Find a relation \( r(X_{vk} \rightarrow X_{vu}) = R \in \text{RT} \)
      where \( X_{vk} \) represents variables in \( r \) belonging to \( X_k \) and \( X_{vu} \) represents variables in \( r \) belonging to \( X_u \), such that size of \( X_{vu} \) is minimum among all \( r \in \text{RT} \). In other words find the relation with fewest numbers of UNKNOWN variables.
   b. Create a bi-partite graph with nodes from \( X_{vk} \) on one side and nodes from \( X_{vu} \) on the other side and arcs from all nodes in \( X_{vk} \) to all nodes in \( X_{vu} \). Make all the UNKNOWN variables in \( r \) depend on KNOWN variables in \( r \). This encodes our intuition that the KNOWN variables in \( r \) would be used to compute values for the UNKNOWN variables in \( r \).
   c. If \( r \) is a local constraint (i.e., \( r \in R_{SL} \)) belonging to location \( l_m \) of component \( c_m \) then add an arc from \( X_{cm} \) to all \( x \in X_{vu} \). This encodes the dependency that a local relation will only be used when the system is in that location.
   d. Move all variables in \( X_{vu} \) from \( X_{vu} \) to \( X_v \) i.e., For each \( x_{vu} \in X_{vu} \), \( X_{vu} = X_{vu} - v_{ui} \) & \( X_{vi} = X_{vi} \cup X_{vu} \). All UNKNOWN variables in \( r \) can now considered to be KNOWN through computation.
   e. Remove \( r \) from the TOBEPROCESSED list. \( R = R - r \).

The conditional probability tables \([P]\) are computed as follows:

1. For all \( x \in X_v \) such that \( x \) corresponds to an input or state variable set \( p(x=\text{CONSISTENT}) = 1.0 \), \( p(x=\text{INCONSISTENT}) = 0.0 \). We will see in the next section that if there are arcs to the state variables from the BN fragment at previous time step then the \( P \) for the state variables will be different.
2. For all other variables (non-input and non-state) \( x \in X_s \) that have incoming arcs only from variables in \( X_v \), \( P \) is set to
   a. \( p(x=\text{CONSISTENT} | \text{for all } x_i \in X_v, x_i = \text{CONSISTENT}) = 1.0 \)
   b. \( p(x=\text{INCONSISTENT} | \text{for all } x_i \in X_v, x_i = \text{CONSISTENT}) = 0.0 \)
   c. \( p(x=\text{CONSISTENT} | \text{there exists } x_i \in X_v, x_i = \text{INCONSISTENT}) = 0.0 \)
   d. \( p(x=\text{INCONSISTENT}) \) there exists \( x_i \in X_v, x_i = \text{INCONSISTENT}) = 1.0 \)

3. For variables \( x \in X_v \) with arcs from other variables \( X_s \) as well as arcs from \( x \). For any variable \( x \in X_v \), there can only be one variable \( x_i \in X_v \) with an arc to \( x \) the \( P \) when \( x_i = l_{\text{current}} \) where \( l_{\text{current}} \) is current location of component \( c \) in SL is set 1 and 0 otherwise.
   a. \( p(x=\text{CONSISTENT} | x_i = l_{\text{current}}) = 0.0 \)
   b. \( p(x=\text{INCONSISTENT} | x_i = l_{\text{current}}) = 1.0 \).

4. For all variables \( x_i \in X_v \), we set the probability of the variable being in the location predicted by the candidate to be 1 and probability of any other location to be 0
   a. \( p(x_i = l_{\text{current}}) = 1.0 \)
   b. \( p(x_i = l_b \text{ for all } l_b \neq l_{\text{current}}) = 0.0 \).

As we will see in the next section, if \( x \) has an arc from \( x_i \) at the previous time step then the \( P \) is computed differently.

### 4.2 Bayesian network across time steps

Once we have generated the BN fragment for candidate \( d_k \) at a specific time step \( t_i \) (\( B_{N_i} \)) we have to connect it to \( B_{N_{k-1}} \) (associated with earlier time steps). Let \( B_{N_{i-1}} = \{X_{n_{i-1}}, \{E_{i-1}, \{P_{n_{i-1}}\}\} \) represent the part of \( B_{N_{k-1}} \) corresponding to the previous time step. We augment \( B_{N_{k-1}} \) as follows:

- First we add a new set of variables \( X_T = \{x_{T1}, x_{T2}, \ldots, x_{Tm}\} \) where \( x_{Ti} \) corresponds to an unguarded transition taken by component \( c_i \). Hence there will be \( n \) such variables where \( n \) is the number of components. The domain for any one of these variables \( x_{Ti} \in X_T \) will be all the unguarded transitions out of \( l_{\text{current}} \) where \( l_{\text{current}} \) is the current location of component \( c_i \) plus an additional transitional called the self transition (\( T_{\text{self}} \)) which if taken keeps the component in the same location.

- Next we add arcs \( E_T \) that represents the conditional dependence of the location of component at \( t_i \) on the transition the component takes between \( t_i-1 \) and \( t_i \). Each arc \( E_t \) is of the form \( x_{Ti} \rightarrow x_{T[t_i]} \).

- Then we add arcs \( E_c \) indicating the conditional dependence of the location of a component at time step \( t_i \) on the location of the same component at time step \( t_i-1 \). Each arc \( E_c \) is of the form \( x_{T[t_i-1]} \rightarrow x_{T[t_i]} \).

- Then we add arcs \( E_{c} \) to represent the conditional dependence of state variables on derivative variables as determined by the Integration model. For each state variable \( x_v \), we determine the set of derivative variable \( X_v \) that will be required to compute the value of \( x_v \). We then add arcs from each \( x_{d} \in X_d \) to \( x_v \).

- The \( P \) for \( x_{T[t_i]} \) is unchanged since it has no new incoming arcs.
- The \( P \) for \( x \in X_T \) is set to the prior probabilities of the corresponding unguarded transitions.
- The \( P \) for \( x_{T[t_i]} \) associated with component \( c \) is computed as follows. Based on the location of \( c \) at \( t_{i-1} \)
and the transition T taken by component between $t_{i-1}$ and $t_i$, we can deduce the location of $c$ at $t_i$. The probability for this location is set to 1 and the probabilities for the rest of the locations are set to 0.

- $p(x_i,t_i|l_i=x_{d_i}[t_{i-1}], l_{i-1}=CONSISTENT, x_{d_i}[t_{i-1}], l_{i-1}=CONSISTENT) = 1.0$
- $p(x_i,t_i|l_i=x_{d_i}[t_{i-1}], l_{i-1}=CONSISTENT, x_{d_i}[t_{i-1}], l_{i-1}=CONSISTENT) = 0.0$

The P for $x_i[t_i]$ for all state variables is updated by setting it to be CONSISTENT when all derivative variables that influence it from the previous time step ($X_{d_i}[t_{i-1}]$) and corresponding state variable from previous time step ($x_i[t_{i-1}]$) are CONSISTENT

- $p(x_i[t_i]=CONSISTENT| X_{d_i}[t_{i-1}], l_i=CONSISTENT) = 1.0$
- $p(x_i[t_i]=CONSISTENT| X_{d_i}[t_{i-1}], l_i=CONSISTENT) = 0.0$
- $p(x_i[t_i]=CONSISTENT) = 1 - p(x_i[t_i]=CONSISTENT)$

### 5 Example and Results

To illustrate the advantage of using the proposed approach, we present a two tank example. The system consists of two tanks with outlet pipes from both tanks and a connecting pipe between the two tanks. A flow source feeds liquid into the first tank. Each of the pipes and the source can be in nominal mode (resistance is specified constant) or in highResistance mode (resistance is 5 times the specified constant). The actual highResistance fault in the system is usually not exactly 5 times but could vary between 4 and 6 times the resistance. The out flows from both outlet pipes are the only observed variables. The sensors associated with the out flows are assumed to have White Gaussian noise. The HyDE model of this example is illustrated in Figure 1.

**Error! Reference source not found.**

In this example, due to feedback effects, all of the faults will eventually impact both observed variables. However the presence of integrating elements in the form of tank capacitances will introduce a delay in the propagation of fault effects. For example, if Pipe1 is in highResistance mode then we should see an immediate influence in the observed Pipe1 outflow while the influence on Pipe2 outflow will take a few time steps to manifest. If we use a purely consistency based diagnosis then we have to wait until both observations have deviated (because no fault influences only one observation) and even then all fault candidates are possible and they will be tested in the order of prior probabilities (Pipe12, Pipe2, Pipe1, Source in that order for this example). The presence of sensor noise and uncertainty about actual magnitude makes it very difficult to absolutely determine if sensor observations deviate from model predictions. If the sensor noise is high or if the actual fault magnitude is not close to 5, the chances of misclassification are quite high.

We ran 34 scenarios by varying the location (4 scenarios) and magnitude of fault (15 scenarios) as well as the sensor noise level (15 scenarios). If a sensor observation was not found to be within 2 standard deviations of a Gaussian distribution (95% of the time values sampled will be within 2 standard deviations) with mean set to the model prediction and standard deviation set based on level of sensor noise then it was considered to be inconsistent. The results are summarized in **Error! Reference source not found.** Figure 3 (Dark Rectangles on the left). We can clearly see that there are a large number of missed alarms and false alarms when using a purely consistency-based approach.

For the second set of experiments we used the proposed modification to HyDE using BN. The BN was automatically generated from HyDE model (one example is illustrated in Figure 2). Probabilities for observed variables were computed using the probability distribution function for Gaussian distribution given by:

$$p(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

**Error! Reference source not found.** Figure 3 (Light Rectangles on the right) shows the results of integrating the BN in the consistency-based diagnosis framework. There is a significant reduction in both missed and false alarms since this approach does not commit to a decision which may later turn out to be erroneous. However we do pay the penalty of using more time and computational resources for the BN computation.
6 Conclusion and Future Work

We presented a new approach for incorporating probabilistic information in a consistency-based diagnosis framework. This approach uses automatically constructed BN models for candidate generation. The conditional probability distributions in the BN are based on probabilistic measures of the consistency between model predictions and observations. BN inference can be used to identify the most likely hypothesis which if different from the original candidate is used to generate a new candidate.

The approach presented in this section differs from typical consistency-based approaches in that it is able to deal with uncertainty when comparing observations and predictions from the model. It differs from typical BN approaches to diagnosis in that it uses BN inference only for the candidate generation rather than entire diagnosis process [Roychoudary et al., 2006].

This new approach combining BN and consistency-based is meant to be useful when a significant amount of uncertainty exists in testing candidates for consistency. In such situations the use of the proposed approach will result in fewer false alarms and missed alarms. Additionally because fewer candidates are likely to be tested, there is an improvement in time and memory performance as well. However in some cases there might be an increase in time and memory performance because of the need to construct the BN and perform inference on it. For the purposes of this paper we used the SAMIAM tool from UCLA (http://reasoning.cs.ucla.edu/samiam/) for Bayesian network inference. In future work we would like to explore the use of strategies to improve the performance of the BN inference including pre-generation and compilation approaches suggested by Darwiche [Chavira and Darwiche, 2007].

Once we have established the framework to integrate BN inference for candidate generation in consistency-based diagnosis there is scope for incorporating several kinds of uncertainty in the reasoning. For example, sensor noise can be modeled directly by the conditional probability distributions for the BN variables corresponding to observable variables. Similarly it is possible to incorporate uncertainty about model parameters in the BN. In future work we would like to explore the extension of BN to support other such forms of uncertainty and see how that improves the sensitivity of the diagnosis algorithm.

As we mentioned earlier this approach can be easily adapted to work with other consistency-based techniques as well. This can be achieved by providing an algorithm to construct the BN from whatever modeling paradigm is being used. For example, the temporal causal graphs in the TRANSCEND system [Mosterman and Biswas, 1999] already provide a structure that can be used to construct the BN. Even more diagnostic power is possibly by changing the BN variable states to (-, 0, +) as used in the TRANSCEND reasoning algorithms. The signal to symbol transformation algorithms have to be modified to output a
probabilistic measure of consistency (or a probability distribution over -, 0, + if those states are used). Using this approach, TRANSCEND would not be forced to commit to a set of candidates generated by the initial backtracking. Alternately it might be possible to start the fault isolation reasoning earlier since it is not essential for 100% accuracy in determining symbols.

Acknowledgments

We would like to thank the HyDE team members Lee Brownston and David Hall for providing useful feedback on the approach discussed in this paper. We would also like to thank Adnan Darwiche’s group at UCLA for use of the SAMIAM tool.

References


[Narasimhan, et al., 2004] Sriram Narasimhan, Richard Dearden, and Emmanuel Benazera. Combining Particle Filters and Consistency-based Approaches for Monitor-
Model-Based Reasoning with Multiple Test Cases and its Application to Debugging

Bernhard Peischl¹,² Naveed Riaz¹ Franz Wotawa¹,²
Technische Universität Graz
Institute for Software Technology
{peischl, nriaz, wotawa}@ist.tugraz.at

Abstract

Today’s simulation-centric hardware development process requires to leverage quality test suites for fault localization rather than employing them solely for detecting malfunctioning. In this article we (1) propose an extension of the model-based debugging theory to address the treatment of test suites in a well-founded way and (2) relate this novel approach to an algorithmic technique known as filtering. For multiple test cases revealing a certain fault (3) we propose an iterative computation of diagnoses as an extension to Reiter’s diagnosis algorithm, and (4) notably report on empirical evaluations of this algorithm taking into account even dual-fault diagnosis.

1 Introduction

Today’s increasing demand on semiconductors and software-enabled systems is accompanied by increasing design complexity, high quality attributes, cost pressure and shrinking time to market. Thus detecting, localizing, and fixing faults is a crucial issue in today’s fast paced and error prone development process. In general, detecting and repairing misbehavior in an early stage of the development cycle reduces costs as well as development time considerably.

Hardware description languages like VHDL [Navabi, 1993; IEEE, 1988] and Verilog [IEEE, 1995], and the availability of mature simulation technology allow for verification and systematic testing in early stage in the development cycle. Unlike to the early 90s, the need for physical prototypes is more and more replaced by appropriate models of the chip under development. Once a fault is detected, the presence of a physical prototype suggests to conduct supplementing measurements for gaining further observations to locate the misbehavior’s root cause. In this scenario, typically the input remains the same, and the engineer looks for further measurement points. The classical literature on model-based diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987] addresses this process and provides foundational support for locating faulty components by incorporating additional measurements.

With the advent of various models in today’s development process, particularly exhaustive simulation of digital semiconductors has become an inherent and well-established technique. Model-based test pattern generation furthermore allows for straightforward generation of test suites fulfilling various coverage goals. However, the availability of these quality test suites to our best knowledge is not addressed by MBSD (model-based software debugging) theory so far. In contrast to the scenario in the early 90s, we need to take advantage of numerous individual test cases rather than capturing additional observations obtained from the prototype. Extending classical model-based debugging theory thus might provide the theoretical underpinning to master the fault location process (which accounts for 50 to 80 percent of the time used for verification depending on the level of automation of the employed verification tools [Auerbach et al., 2005]) and promises to provide remedy in terms of sophisticated tool support.

In this article we briefly introduce MBSD and (1) propose to extend MBSD to leverage whole test suites rather than individual tests (see Section 3). Moreover, (2) we relate this approach to the previously proposed method of filtering (also see Section 3) and (3) present a novel extension to Reiter’s hitting set algorithm that efficiently takes account of multiple error-revealing test cases (see Section 4). We (4) outline empirical results (notably single- as well as dual-fault diagnosis) obtained from the ISCAS’89 benchmark suite (see Section 5). Finally we discuss related literature (see Section 6), and conclude this article (see Section 7).

2 Model-based Software Debugging

The basic idea of model-based software debugging (MBSD) is to employ a single test case together with knowledge about the program’s syntax and semantics to locate the misbehavior’s possible causes. Obviously, a correctly functioning program cannot produce an incorrect value for a given test case. Therefore, to make the program consistent with this specific test case revealing the faulty behavior, we have to assume
some subset of the program’s components to work incorrectly. We report these components as diagnosis candidates for the specific test case under consideration.

MBSD, as the discipline of applying model-based diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987] to locate bugs in software, has a well-founded theory and several case studies indicate its maturity in the context of HDLs [Peischi and Wotawa, 2006; Wotawa, 2002]. In the following we restate the definition of the classical diagnosis problem [Reiter, 1987; de Kleer and Williams, 1987] and point out the most notable differences to MBSD.

Definition 2.1 (Diagnosis Problem) [Reiter, 1987; de Kleer and Williams, 1987] A diagnosis system is a pair \( (SD, COMP) \), where

- \( SD \), the system description, in [Reiter, 1987] is a set of first-order sentences
- \( COMP \), the system components, is a finite set of constants.

An observation of a system is a finite set of first-order sentences, and together with the system description \( (SD, COMP) \) forms the classical diagnosis problem \( (SD, COMP, OBS) \).

In contrast to a diagnosis problem, where \( SD \) specifies the correct behavior and the observations specify the input and the response of the actual system, the system description \( SD \) of a debugging problem describes the (faulty) behavior and the tests (partially) specify the intended (correct) behavior. Moreover, to overcome scalability issues, we employ a Horn-clause like encoding for both, \( SD \) and the test cases.

Definition 2.2 (Debugging Problem) A debugging problem is a tuple \( (SD, COMP, TC) \) where \( SD \) is a logical model of the given program (typically incorporating structure and behavior), \( COMP \) is a finite set of statements or expressions of this program, and \( TC \) refers to a logical sentence representing the given test case.

3 Test Suites for Fault Localization

Although some research work considers multiple test cases (or test suites) for enhancing the proposed model’s discrimination capability in terms of filtering the diagnosis obtained from a specific model [Wotawa, 2002], we notably lack a general approach for the treatment of whole test suites. In the following we extend the MBSD framework towards the universal treatment of test suites. Within this framework we establish a relationship to the filtering approach presented in [Wotawa, 2002].

Rather than considering a single test case solely, a test suite contains multiple test cases \( TC_1, TC_2, ..., TC_n \), while the system description remains the same. As outlined in Section 1, test cases may either reveal a fault - that is, the specific test case’s logical encoding is inconsistent with the system description, or fail to detect misbehavior. For the remainder of this article, the first case refers to a negative test case, whereas we refer to a positive test in case of consistency. Thus we can partition the set of test cases \( TC \) in positive (referred to as \( TC_{pos} \) in the following) and negative ones (formally referred to as \( TC_{neg} \)).

We continue with a simple example illustrating the potential of positive test cases.

Example 3.1 (Positive test cases) Figure 1 illustrates a part of a circuit comprising an XOR and a NOT gate. We further assume that this circuit is faulty and that both components are reported as diagnosis candidates by employing negative test cases. Suppose we have already received the following information after the application of a negative test case \((in1 = '1', in2 = '0', out = '1')\). Considering the NOT gate abnormal and XOR gate correct, the value of ‘inter’ is computed ‘0’ by the model. Now we apply a positive test case \((in1 = '0', in2 = '0', out = '1')\). Once again considering the NOT gate abnormal and XOR gate correct, the value of ‘inter’ is now computed ‘1’ by the model. We immediately see that abnormal component NOT is required to map ‘inter’ = ‘0’ for negative test case and ‘inter’ = ‘1’ for the positive test case for the same input value \( in2 = '0' \). Obviously, no deterministic component can fulfill this requirement. Thus the non-component can no longer be considered as a valid diagnosis candidate.

In the following we generalize the idea motivated by our example and show how to incorporate positive test cases into the MBSD framework. Afterwards we relate our novel system description to the algorithmic filtering approach presented in [Wotawa, 2002].

Definition 3.1 (Test Suite Integration) Given a set of test cases \( TC = TC_1, TC_2, ..., TC_n \), a system description \( SD \) and a set of components, the diagnosis problem considering all test cases in \( TC \) is obtained as follows:

- for each \( TC_i \in TC \) do
  - generate a new \( SD_i \), where all component and connections are uniquely identified with a new index \( i \)
  - let \( SD^* = \bigcup_{i=1}^{n} SD_i \cup \{\neg AB(C) \rightarrow \neg AB(C_1) \land \neg AB(C_2) \land ... \land \neg AB(C_k) | C \in COMP\} \)
- let \( TC^* \) be a renaming of \( TC \), such that every test case is associated with connections in \( SD_i \)

The new diagnosis problem incorporating the test cases \( TC \) is given by the tuple \( (SD^*, COMP, TC^*) \).

Note that the size of \( SD \) increases with the number of test cases linearly and the individual components \( C_{i, i = 1, 2, ..., k} \) are treated as independent components. Applying Reiter’s algorithm [Reiter, 1987] in a straightforward way.
is thus rather inefficient. In Section 4 we thus propose a novel, iterative construction of the hitting-set DAG. However, although this novel algorithm handles additional conflicts rather efficient, positive test cases do not yield to further conflicts and thus are not able to discriminate diagnosis further on. Under absence of structural faults, the following, novel extension allows even for taking advantage of positive test cases by relying on Ackermann constraints [Ackermann, 1954].

As positive test cases do not yield to additional conflicts, we capture their specific information on diagnoses in terms of the system description with Ackermann constraints $SD^A$. To our best knowledge the authors of [Raiman et al., 1991] were the first employing Ackermann constraints to express that our components behave deterministically. By adding these consistency constraints we formalize the fact that the same combination of input values applied to a component $C$ produces the same output for every instance $C_i$. This specifically allows for exploiting valuable information captured in terms of the many test cases not revealing any faulty behavior.

**Definition 3.2 (System Descr. with Ackermann constraints)**

Given a set of positive test cases $TC_{pos}$, we assume $in(C_i) = \{i_1, \ldots, i_m\}$ to denote the inputs of component $C_i$, and $out(C_i) = \{o_1, \ldots, o_k\}$ the outputs of component $C_i$. By extending the system description $SD^*$ in terms of the Ackermann Constraints $\neg AB(C) \land CON_A = \{\forall_{p=1}^m i_{c_p} = o_{c_p}\} \land \neg \exists i, j \neq j, i \neq j, (\forall_{p=1}^m i_{c_p} = o_{c_p})$, where $i$ and $j$ denote indices of test cases, we obtain a diagnosis problem incorporating Ackermann constraints. The diagnosis problem is thus given in terms of the tuple $(SD^A, COMP, TC_{pos}^*)$ where $SD^A = SD^* \cup CON_A$ denotes the Ackermann system description.

An algorithmic approach to these constraints is filtering [Wotawa, 2002]. Filtering refers to discarding certain diagnosis by taking advantage of further test cases $TC_i$ in a dedicated post processing phase. A diagnoses $\Delta$ states that $\Delta \cup SD \cup TC_i \cup \neg AB(C) \mid C \in COMP \setminus \Delta$ is consistent. This implies that there is a replacement, that is - there exists a function $replace(C)$ for every component $C \in \Delta$ - replacing the program for the given test case $TC_i$.

We briefly restate filtering as follows. Since all components in $COMP \setminus \Delta$ are assumed to behave correctly, we can compute the input values $in(C)$ and output $out(C)$ for every component $C \in \Delta$ (Using simulation). According to this computed input-output relation obtained from all test cases $TC_i$, component $C$ is possibly required to map the same input to different output values. This corresponds to an inconsistency and the specific diagnosis $AB(C)$ is not repairable w.r.t the test cases $TC_i$. As there is no function $replace(C)$ as stated previously component $C$ can be removed from the set of diagnosis candidates. In this vein, we basically evaluate the Ackermann constraints in an iterative fashion at a meta-model level by checking for different input values for a certain output value.

We formalize the procedure mentioned previously in terms of the procedure $filter(\Delta, TC)$, where $\Delta$ denotes the set of diagnosis candidates and $TC$ represents the test suite:

1. forall test cases $TC_i \in TC$ do
2. (a) forall $D \in \Delta$ do
   (b) Let $i_d$ denote the values at the input and $o_d$ be the values at the output of component $D$ obtained by assuming $AB(D) \land \{\neg AB(C) \mid C \in COMP \setminus \Delta\}$
   (c) if there exist indices $i, j, i \neq j$, such that $i_d = i_d, o_d \neq o_d$
   (d) remove $D$ from $\Delta$
3. return $\Delta$

**Claim 1 (Proof of Claim 1)** The procedure $filter(\Delta, TC)$ reduces the diagnosis candidates $\Delta$ obtained from $(SD, COMP, TC)$ exactly to those given in terms of the constraints problem $(SD^A, COMP, TC^*)$, where $SD^A$ again denotes the system description $SD$ with Ackermann constraints as given in Definition 3.2 and $TC^*$ refers to the renaming as given in Definition 3.1.

**Proof of Claim 1** After applying $filter(\Delta, TC)$ to the obtained diagnoses, there is no component $D$ at which we obtain different input values for a certain output. Following the notion given in Definition 3.2, we formally conclude
1. $\forall i, j, i \neq j \cdot (\forall_{p=1}^m i_{c_p} = o_{c_p}) \land \forall_{p=1}^m o_{c_p} \neq o_{c_p}$
2. $\exists i, j, i \neq j \cdot (\forall_{p=1}^m i_{c_p} = o_{c_p}) \land \forall_{p=1}^m o_{c_p} \neq o_{c_p}$
3. $\forall i, j, i \neq j \cdot (\forall_{p=1}^m i_{c_p} = o_{c_p}) \land \forall_{p=1}^m o_{c_p} \neq o_{c_p}$
4. $\forall i, j, i \neq j \cdot (\forall_{p=1}^m i_{c_p} = o_{c_p}) \land \forall_{p=1}^m o_{c_p} \neq o_{c_p}$
5. $\forall i, j \cdot CON_A$

The procedure $filter(\Delta, TC)$ thus imposes the Ackermann constraints on the set $\Delta$.

4 Negative Test Cases: Iterative Computation of Diagnosis

For negative test cases we rely on reusing Reiter’s well known Minimal HITting Set Tree Algorithm [Reiter, 1987] for computing diagnosis. This algorithm was later revised by Greiner [Greiner et al., 1989]. First we generate a Minimal HITting Set Directed Acyclic Graph (referred to as HS-DAG in the following) for the first test case (similar to actual Greiner’s algorithm for single test case). The nodes labeled by $\triangledown$ represent the minimal diagnosis for current test case. Further test cases traverse, reuse and extend the same directed acyclic graph.

This directed acyclic graph is traversed using breadth first strategy and further test cases modify this acyclic directed graph by adding new nodes or closing, pruning or reusing old nodes. The main idea is that, first an acyclic directed graph $D$ is created, the nodes which represent minimal diagnosis and are labeled by $\triangledown$ are replaced with conflict sets $CS$ returned from next test cases, if such conflict sets exist. If no such
conflict set exists then this node also represents the minimal
diagnosis for next test cases as well. $H(\sqrt{ })$ is the set of edge
labels on the path from root down to node labeled by $\sqrt{ }$.

We use a simple example to illustrate our approach. Suppose we have two test cases, $TC_1$ and $TC_2$. $F_1 = \{\{1,2,3\},\{1,3\},\{1,4\}\}$ and $F_2 = \{\{1,4,5\},\{3,4\},\{1,2\}\}$ denote the corresponding conflict sets.

Figure 2 represents their corresponding minimal HS-DAG.

![Figure 2: MHS directed acyclic graphs for TC1 and TC2](image)

Suppose $D_1 = \{\{1\},\{3,4\}\}$ and $D_2 = \{\{1,3\},\{1,4\},\{2,4\},\{2,3,5\}\}$ are diagnoses returned from acyclic directed graphs obtained from $F_1$ and $F_2$ respectively.

By following the well-known procedure proposed in [Greiner et al., 1989] we obtain a DAG $DAG_{TC_1}$, as depicted in Figure 2. For incorporating $F_2$, we start traversing this DAG in breadth first order. We find node $n_1$, marked with $\sqrt{ }$ at level 1 with $H(n_1) = \{1\}$. We can see that this node can be labeled with conflict set $\{3,4\}$ of $F_2$, so we replace the label $\sqrt{ }$ of $n_1$ with $\{3,4\}$ and create two downwards arcs with labels 3 and 4 respectively.

Now we have created two new unlabeled nodes $n_6$ and $n_7$ with $H(n_6) = \{1,3\}$ and $H(n_7) = \{1,4\}$. In the same level there is no other node which can be re-processed so we move to next level.

We have unlabeled nodes $n_6$ and $n_7$. As there is no conflict set available in $F_2$ which can become their label, these are labels 3 and 4 respectively.

Figure 3: $DAG_{mult}$, the DAG generated by our novel alg.

![Figure 3](image)

The new minimal node $n_6$ is at the same level as $n_4$, so we can reuse $n_6$.

Moving forward we find node $n_5$ previously marked with label $\sqrt{ }$ and $H(n_5) = \{3,4\}$. Now this label can be replaced with conflict set $\{1,2\}$ of $F_2$, so we replace the label $\sqrt{ }$ of $n_5$ with $\{1,2\}$. On the next level we have two unlabeled nodes represented as $n_8$ and $n_9$ and $H(n_8) = \{1,3,4\}$ and $H(n_9) = \{2,3,4\}$. As we already have minimal node $n_6$ with $H(n_6) = \{1,3\}$ so we can close the first node. For the second node, as there is no conflict set available in $F_2$ which can become it’s label, we mark it with label $\sqrt{ }$. Our final DAG is depicted in Figure 3. We get the following diagnosis from these two test cases $D_{mult} = \{1,3\},\{1,4\},\{2,3,4\}$.

4.1 Algorithm

To efficiently treat multiple test cases, we follow Greiner’s original algorithm [Greiner et al., 1989] constructing the HS-DAG for an ordered collection of sets $F_i$. We assume to use the same pruning rules, i.e., node closing, node re-use, and node pruning. The latter is used in cases where a conflict set is found which is a subset of a conflict set used earlier in the construction of the HS-DAG. All pruning rules except node closing remain the same in our variant of Reiter’s HS-DAG algorithm.

Node closing is used in cases where a node $n$ is pro-
cessed and there exists another node $m$ labeled with $\sqrt{ }$ and $H(m) \subset H(n)$. In this case $m$ already reveals a minimal hitting-set and $n$ would only lead to a non-minimal one. Thus $m$ can be closed which is indicated by labeling $m$ with $\times$. When changing a HS-DAG incrementally using multiple test cases a previously generated minimal hitting set might be-
come invalid because a new previously not considered conflict is detected. Hence, the corresponding node $n$ has to be extended and its label has to change. This of course has consequences for nodes $m$ which has been closed because of $n$. In order keep track of closed nodes $m$, we introduce a new function $closed$ for nodes $n$ which stores all nodes that are closed because of $n$. Hence, when changing the label of $n$ from $\sqrt{ }$ to a conflict set, we immediately know the closed nodes we have to re-process.

The following iterative version of Reiter’s HS-DAG algo-

rithm assumes that we have given a set $F$ which comprises
set of conflicts $F_i$, $i = 1,..,n$. For each test case $TC_i$ the
set $F_i$ stores the conflicts obtained from $TC_i$ and the system
description. Note that it is not necessary to compute all con-
licts for given test case in advance. They can be computed
whenever required. We use the same technique described by
Reiter [Reiter, 1987] for this purpose.

1. Construct the root node $n_0$ of $D$. Let $H(n_0)$ be the
empty set and label the node with $\sqrt{ }$.
2. For each element $F_i$ from $F$ do the following:
   (a) Traverse $D$ in breadth first order starting from the
   root node $n_0$.
   (b) if node $m$ is previously labeled by $\sqrt{ }$
      i. If for all $x \in F_i$, $x \cap H(m) \neq \{\}$ then leave this
         node as it is and move to next node.
      ii. Else label $m$ by $\sum_i$ where $\sum_i$ is the first mem-
          ber of $F_i$ for which $\sum_i \cap H(m) = \{\}$ and for

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.
each $\sigma \in \sum$, generate a new downward arc labeled by $\sigma$. This arc leads to a new node $o$ with $H(o) = H(m) \cup \{\sigma\}$. The new node $o$ will be processed (labeled and expanded) after all nodes in the same generation as $m$ have been processed. Re-open all nodes $n \in closed(m)$ by removing their label.

(c) If a node $m$ is unprocessed, i.e., its label is empty, do the following:

i. If for all $x \in F_i : x \cap H(m) \neq \{\}$ then label $m$ by $\sqrt{}.$

ii. Else, label $m$ by $\sum$ where $\sum$ is the first member of $F_i$ for which $\sum \cap H(m) = \{\}$. For each $\sigma \in \sum$, generate a new downward arc labeled by $\sigma$. This arc leads to a new node $o$ with $H(o) = H(m) \cup \{\sigma\}$. The new node $o$ will be processed (labeled and expanded) after all nodes in the same generation as $m$ have been processed.

3. Return $D$.

After all test cases in $F$ have been processed the HS-DAG $D$ comprises all minimal hitting sets which are the nodes labeled by $\sqrt{}.$ The pruning rules of Greiner et al. are used in order to ensure $D$ to be as small as possible. Moreover, because of the breadth first computation the algorithm computes hitting sets of increasing size. We might stop computation for hitting sets exceeding a given limit.

5 Empirical Results

In this section we evaluate our approach using circuits from ISCAS 89 benchmark suite [Brglez et al., 1989]. For every circuit we introduced a fault by substituting a randomly selected statement with another one, e.g., changing an and operator with an or operator. We obtained the error revealing-inputs - the negative test cases - by performing a circuit equivalence check with the model checker VIS [R. K. Brayton et al., 1996]. Whenever we invoke VIS, we obtain a different negative test case for the specific circuit. To supplement our results on the filtering approach [Wotawa, 2002], we focus our empirical research work on negative test cases.

Table 1 outlines the obtained results for the first 8 circuits in the ISCAS 89 benchmark: In column one the table lists the circuit name, column two shows the length of the error-revealing input sequence, and column three refers to the number of components building up the model. Column four opens a sub-table which, for a number of test cases (ranging from one to five for every circuit), lists the number of fault candidates, the number of faulty lines (a single source may correspond to several faults), and the percentage of reduction (last column) in terms of source code lines. The last column indicates, that under presence of five error-revealing test cases, the proposed extension of Reiter’s algorithm allows for excluding 94 percent (considering 4 cycles) respectively 93 percent (considering 8 cycles) of all source lines. For every individual circuit, we verified that the introduced fault is among the reported ones.

As pointed out in Section 4 our algorithm extension allows for retrieving diagnosis in increasing order of cardinality. Similar to Table 1, Table 2 outlines our most recent results for dual-fault diagnosis (alongside with the total number of source code lines for every circuit). Notably, for functional faults, whilst guaranteeing that the introduced fault is among the diagnosis candidates, our algorithm allows for excluding more than 92 (for single-fault diagnosis) and 85 percent (for dual-fault diagnosis) of the source code considering up to at most five negative test cases.

Due to semantic differences between Verilog and VHDL, these results for Verilog are slightly weaker than the results for VHDL RTL presented in [Peischl and Wotawa, 2006]: Our Verilog model differs from the VHDL approach in an additional evaluation component to reflect the correct semantics of Verilog’s blocking and non-blocking assignments [Peischl et al., 2008]. For the ISCAS 89 test suite, on average, 187 of these evaluation components degrade the model’s discrimination capabilities.

![Figure 4: Results ISCAS89, multiple test cases, single fault, 4 (above) and 8 cycles (below)](image)

Note that - particularly in a practical setting - negative test cases may provide different discrimination capabilities with respect to the obtainable diagnoses. Regarding our experiments, the circuit equivalence checking approach solely guarantees to reveal the introduced fault (constraining the test sequence generation by, for example, requiring a test case to affect disjoint output signals - might further excel the presented results).
### Table 1: Empirical results single fault diagnosis, multiple test cases, ISCAS 89 benchmark suite

<table>
<thead>
<tr>
<th>circuit</th>
<th>cyc. no.</th>
<th>comp. no.</th>
<th>fault candid.</th>
<th>faulty lines</th>
<th>detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2360</td>
<td>105</td>
<td>64</td>
<td>73.3</td>
<td>8.8</td>
</tr>
<tr>
<td>3</td>
<td>105</td>
<td>64</td>
<td>73.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>75</td>
<td>50</td>
<td>78.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>75</td>
<td>50</td>
<td>78.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5720</td>
<td>121</td>
<td>130</td>
<td>45.8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>171</td>
<td>102</td>
<td>57.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>134</td>
<td>80</td>
<td>66.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>64</td>
<td>73.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>83</td>
<td>50</td>
<td>78.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3748</td>
<td>20</td>
<td>13</td>
<td>97.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>13</td>
<td>97.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>13</td>
<td>97.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>13</td>
<td>97.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7496</td>
<td>102</td>
<td>122</td>
<td>71.8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>202</td>
<td>122</td>
<td>71.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>53</td>
<td>87.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>17</td>
<td>96.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>11</td>
<td>97.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>11</td>
<td>97.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3688</td>
<td>12</td>
<td>18</td>
<td>98.6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>18</td>
<td>96.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>18</td>
<td>96.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>18</td>
<td>96.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7376</td>
<td>149</td>
<td>82</td>
<td>85.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>139</td>
<td>82</td>
<td>85.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>118</td>
<td>70</td>
<td>87.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>107</td>
<td>64</td>
<td>88.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>39</td>
<td>92.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>26</td>
<td>93.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3904</td>
<td>56</td>
<td>33</td>
<td>91.6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>12</td>
<td>97.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>12</td>
<td>97.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>12</td>
<td>97.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7808</td>
<td>172</td>
<td>75</td>
<td>86.3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>134</td>
<td>75</td>
<td>86.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>28</td>
<td>94.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>34</td>
<td>19</td>
<td>96.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>19</td>
<td>96.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>19</td>
<td>96.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3568</td>
<td>198</td>
<td>145</td>
<td>71.5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>251</td>
<td>145</td>
<td>71.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>251</td>
<td>145</td>
<td>71.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7136</td>
<td>231</td>
<td>141</td>
<td>72.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>224</td>
<td>141</td>
<td>72.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>151</td>
<td>96</td>
<td>81.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>151</td>
<td>96</td>
<td>81.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>151</td>
<td>96</td>
<td>81.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4488</td>
<td>104</td>
<td>38</td>
<td>93.1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>12</td>
<td>98.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>12</td>
<td>98.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>193</td>
<td>106</td>
<td>82.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>108</td>
<td>57</td>
<td>90.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>67</td>
<td>36</td>
<td>94.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>43</td>
<td>23</td>
<td>96.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>43</td>
<td>23</td>
<td>96.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4928</td>
<td>185</td>
<td>99</td>
<td>85.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>167</td>
<td>89</td>
<td>86.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>166</td>
<td>89</td>
<td>86.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>166</td>
<td>89</td>
<td>86.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9856</td>
<td>257</td>
<td>214</td>
<td>67.6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>491</td>
<td>257</td>
<td>61.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>399</td>
<td>214</td>
<td>67.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>299</td>
<td>160</td>
<td>73.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>273</td>
<td>145</td>
<td>78.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>189</td>
<td>99</td>
<td>85.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4844</td>
<td>40</td>
<td>20</td>
<td>95.9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>15</td>
<td>97.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>15</td>
<td>97.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9688</td>
<td>22</td>
<td>18</td>
<td>97.1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>22</td>
<td>96.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>18</td>
<td>97.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>17</td>
<td>97.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>31</td>
<td>17</td>
<td>97.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>15</td>
<td>97.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>4</td>
<td>3941</td>
<td>74</td>
<td>44</td>
<td>93.6</td>
</tr>
<tr>
<td>Average</td>
<td>8</td>
<td>7882</td>
<td>123</td>
<td>70</td>
<td>92.5</td>
</tr>
</tbody>
</table>
6 Related Work

We can divide related work into four areas. First we discuss and compare the different debugging techniques used for sequential circuits. Second, we point out work related to treatment of multiple test cases. Third, we discuss the different approaches for diagnosing multiple faults and finally we discuss the work related to Ackermann constraints.

Pitchumani, Mayor, and Radia describe a diagnosis tool for VHDL that employs so-called functional fault models and reasons from first principles by means of constraint suspension [V. Pitchumani et al., 1991; 1992]. They employ a hierarchical approach using stuck-at fault modes at the first level and an arbitrary failure model at the second level. As the authors do not provide experimental results, it is impossible to evaluate whether their approach outperforms ours in terms of no. of fault candidates. Kuchcinski et al. [Krzysztof et al., 1993] discuss an application of algorithmic debugging to automatic fault localization in VLSI designs and propose a smooth combination of different diagnosis techniques.

Our empirical results differ from other published results [Cheng et al., 1998; Alexander et al., 2004] mainly in two points. We perform evaluation on unoptimized RTL representations and we compute possible fault locations at the expression level rather than at the gate level. Authors of [Cheng et al., 1998] also present experimental results for the ISCAS 89 benchmark suite. The satisfiability-based approach [Alexander et al., 2004] also employs optimized versions of the benchmarks and leverages recent achievements in Boolean satisfiability techniques for computing diagnoses. Notably, the authors of that approach point out that an unoptimized version makes diagnosis harder because of circuit redundancies. In source-level debugging, we inherently deal with unoptimized representations because any optimization might remove redundancies and possible fault candidates. Moreover, these researchers claim that fault-mode-free diagnosis is a desirable characteristic because fault effects can have non-deterministic behavior.

Second, we discuss work related to treatment of multiple test cases. Reiter [Reiter, 1987] discusses the concept of incorporating additional measurements and Hou [Hou, 1991] explores this concept further on. F.Levy [F.Levy, 1992] has also described an approach for incremental treatment of different conflict sets. Moreover, the authors of [Meerwijk and Priest, 1992] recognized this lacking aspect and propose an approach for employing multiple test cases, however, their approach imposes unreasonable assumptions on the relationship between the circuit’s input and output.

Third, there have been different approaches taken by different researchers for the diagnosis of multiple faults. The approach in [de Kleer and Williams, 1987] is based on conflict recognition and candidate generation. [H.T.Ng., 1990] provided the extension for the consistency-based diagnosis approach described by Reiter [Reiter, 1987] for the diagnosis of devices whose behavior changes over time. [Subramanian and R.J.Mooney, 1996] presented a similar approach, they combined this standard diagnostic approach [de Kleer and Williams, 1987] with a hypothesis checker. The authors of [Daigle et al., 2006] use a fault isolation scheme, where fault effects are represented as qualitative fault signatures. As we are not aware of any publications reporting on empirical results, on the proposed techniques, it is impossible to compare our approach with their approach in terms of effectiveness.

Finally, To our best knowledge the authors of [Raiman et al., 1991] were the first employing Ackermann constraints to express that components behave deterministically. We pursue a similar idea in the context of positive test cases in a simulation-driven development process. Notably [Staber et al., 2006] employed Ackermann constraints for locating faults in Verilog programs by using a model checker.

7 Conclusion

Today’s simulation-centric hardware development process requires to leverage quality test suites for locating a misbehavior’s cause rather than solely detecting it. In contrast to the early 90s, we therefore need to take advantage of numerous individual test cases rather than narrowing the fault location in terms of additional measurements on a prototype.

Regarding typical test suites only a fraction of these test cases effectively reveals a fault. The remaining ones - although carrying valuable diagnosis information - do not contribute to locate the misbehavior.

This article contributes to MBD research in (1) proposing an extension to the classical model-based diagnosis theory to address the treatment of test suites, (2) relates this extension to a previously proposed filtering approach, (3) shows how to iteratively extend Reiter’s algorithm to leverage multiple error-revealing faults, and (4) reports on an concrete debugging application in terms of an empirical evaluation notably taking into account dual-fault diagnoses.

References


1 Introduction

Providing models representing physical systems is a common concern spread over all scientific and engineering communities. Model based methods require such modeling and rely on the soundness of the models. This is particularly true for model based fault detection and diagnosis. But complex systems are often subjected to uncertainties that make the modeling task awkward. This is why pure numerical models are sometimes disregarded to the benefit of set-membership models which naturally cope with uncertain and inaccurate knowledge.

At some stage of the process, one may face two kinds of uncertainties. On one side, unstructured uncertainties mean that deriving a complete equational model for the physical phenomena is impossible. On the other side, when the structure of the equations is known but some of the parameters are not, uncertainties are said to be structured. In addition to these uncertainties, it is not always possible to get informations about disturbances and noises acting on the system. This may turn the usual stochastic framework inappropriate. In such cases, assuming bounded uncertainties may be a solution.

Considering structured uncertainties, an interesting way to go is to use guaranteed estimation methods, which learn the state and/or parameters of the models from data. These methods rely on interval analysis that first appeared in [Moore, 1966]. They are now subject of a growing interest in various communities and are applied to many tasks [Alamo et al., 2005; Armengol et al., 2001; Guerra et al., 2006; Jaulin et al., 2001; Kieffer and Walter, 1998; Kieffer et al., 2002; Lesecq et al., 2003; Ribot, 2006; Ribot et al., 2007].

This paper presents a fault detection method using interval parameter estimation. Parameters of the model are estimated from the input and output measurements of the system. The consistency of this estimation is then checked against parameters computed from a theoretical (possibly faulty) model of the system. Computations use the set inversion algorithm SIVIA [Jaulin and Walter, 1993; Jaulin et al., 2001]. The results are approximated but are bounded in a guaranteed way. The method is then tested to detect Oscillatory Failure Cases (OFC) in Electrical Flight Control System (EFCS) of civil airplanes, which is a real case study provided by AIRBUS.

The paper is organised as follows. Section 2 presents an overview of interval analysis, its original purpose and its application to fault detection. The error bounded context is then presented more precisely with parametric estimation using intervals in section 3. In section 4, the case study is presented; we describe what are OFC, and their consequences on the aircraft control surfaces, why such failures must be detected in time and one of the methods currently used on Airbus aircrafts for OFC detection. In section 5 the application and the obtained results are analyzed. Finally some conclusions are outlined in last section.

2 Interval analysis

2.1 Preamble

The key idea of interval analysis is to reason about intervals instead of real numbers and boxes instead of real vectors. The first motivation was to obtain guaranteed results from floating point algorithms and it was then extended to validated numerics [Moore, 1959]. Let us recall that in computers real numbers can only be represented by a floating point approximation, hence introducing a quantification error. A guaranteed result means first that the result set encloses the exact solution. The width of the set, i.e. the result precision, may be chosen depending on various criteria among which response time or computation costs. Secondly, it also means that the algorithm is able to conclude on the existence or not of a solution in limited time or number of iterations. The first sig-
significant work is due to Moore in his PhD thesis which was the early beginnings of his reference book [Moore, 1966].

2.2 Main concepts

The matter is to wrap the sets of interest into boxes or union of boxes for which computations may be easier. There are some fundamental operations on intervals which are briefly explained after the definition of an interval.

Interval

A real interval $[u] = [u_0, u_1]$ is a closed and connected subset of $\mathbb{R}$ where $u_0$ represents the lower bound of $[u]$ and $u_1$ represents the upper bound. The width of an interval $[u]$ is defined by $w(u) = u_1 - u_0$ and its midpoint by $m(u) = (u_0 + u_1)/2$.

The set of all real intervals of $\mathbb{R}$ is denoted $\mathbb{I}$. Two intervals $[u]$ and $[v]$ are equal if and only if $u_0 = v_0$ and $u_1 = v_1$. Real arithmetic operations are extended to intervals [Moore, 1966].

Arithmetic operations on two intervals $[u]$ and $[v]$ can be defined by:

\[ \forall \in \{+,-,*,/\}, [u] \circ [v] = \{ x \circ y \mid x \in [u], y \in [v] \}. \]

An interval vector (or box) $[X]$ is a vector with interval components and may equivalently be seen as a cartesian product of scalar intervals:

\[ [X] = [x_1] \times [x_2] \times \ldots \times [x_n]. \]

The set of $n$-dimensional real interval vectors is denoted by $\mathbb{I}^n$.

An interval matrix is a matrix with interval components. The set of $n \times m$ real interval matrices is denoted by $\mathbb{I}^{n \times m}$. The width $w(.)$ of an interval vector (or of an interval matrix) is the maximum of the widths of its interval components. The midpoint $m(.)$ of an interval vector (resp. an interval matrix) is a vector (resp. a matrix) composed of the midpoint of its interval components.

Classical operations for interval vectors (resp. interval matrices) are direct extensions of the same operations for punctual vectors (resp. punctual matrices) [Moore, 1966].

Inclusion function

Given $[u]$ a box of $\mathbb{I}^n$ and a function $f$ from $\mathbb{I}^n$ to $\mathbb{I}^m$, the inclusion function $f$ aims at getting an interval containing the image of $[u]$ by $f$.

The range of the function $f$ over $[u]$ is given by:

\[ f([u]) = \{ f(x) \mid x \in [u] \}. \]

The interval function $[f]$ from $\mathbb{I}^n$ to $\mathbb{I}^m$ is an inclusion function for $f$ if:

\[ \forall [u] \in \mathbb{I}^n, f([u]) \subseteq [f([u])]. \]

An inclusion function of $f$ can be obtained by replacing each occurrence of a real variable by its corresponding interval and by replacing each standard function by its interval evaluation. Such a function is called the natural inclusion function. In practice the inclusion function is not unique, it depends on the syntax of $f$.

Inclusion test

Given a subset $S$ of $\mathbb{R}^n$, we test if $[x]$ belongs to $S$, more precisely if $[x] \subseteq S$ or $[x] \cap S = \emptyset$. These tests are used to prove that all points in a given box satisfy a given property or to prove that none of them does.

Contractor

The last operation is the contraction of $[x]$ with respect to $S$. This means that we search a smaller box $[z]$ such that $[x] \cap S = [z] \cap S$. If $S$ is the feasibility set of a problem and $[z]$ turns out empty, then the box $[x]$ may not contain the solution [Jaulin et al., 2001].

These operations are used to test if a box can or cannot be removed from the solution set. When no conclusion can be drawn, the box may be bisected and each of the sub-boxes can be tested in turn (this corresponds to branch-and-bound algorithms).

2.3 SIVIA: Set Inversion Via Interval Analysis

Consider the problem of determining a solution set for the unknown quantities $u$ defined by

\[ S = \{ u \in U \mid \Phi(u) \subseteq [y] \}, \]

where $[y]$ is known a priori, $U$ is an a priori search set for $u$ and $\Phi$ a nonlinear function not necessarily invertible in the classical sense. (1) involves computing the reciprocal image of $\Phi$. This can be solved using the algorithm SIVIA, which is a recursive algorithm that explores all the search space without losing any solution. This algorithm makes it possible to derive a guaranteed enclosure of the solution set $S$ as follows:

\[ S \subseteq S \subseteq \mathbb{S}. \]

The inner enclosure $\mathbb{S}$ is composed of the boxes that have been proved feasible. To prove that a box $[u]$ is feasible it is sufficient to prove that $\Phi([u]) \subseteq [y]$. Reversely, if it can be proved that $\Phi([u]) \cap [y] = \emptyset$, then the box $[u]$ is unfeasible. Otherwise, no conclusion can be reached and the box $[u]$ is said undetermined. The latter is then bisected in two sub-boxes that are tested until their size reaches a user-specified precision threshold $\varepsilon > 0$. Such a termination criterion ensures that SIVIA terminates after a finite number of iterations.

2.4 Fault detection using intervals

Set membership detection uses previous concepts to perform state estimation and parameters estimation. In state estimation, a nonlinear dynamical model is approximated by a Taylor expansion [Rihm, 1994; Berz and Makino, 1998; Nedialkov et al., 2001] to compute a box enclosing all possible trajectories of the solution between two successive time steps $t_f$ and $t_{f+1}$.

The fixed point and Picard-Lindelöf theorems prove the existence and uniqueness of the solution [Rihm, 1994]. The interval solution becomes obviously wider and wider at each iteration step: this drawback is known as the wrapping effect. Numerous methods may circumvent this pessimism: among them one is to use high order Taylor expansion, mean value forms, matrices preconditioning and a predictor-corrector approach [Corliss, 1994; Nedialkov, 1999; Neumaier, 1990; Raissi et al., 2004; Ramdani, 1995; Rihm, 1994].
3 Parameter estimation in a bounded error context

Parameters and state estimation from experimental measures are usually obtained within a stochastic framework in which known distribution laws are associated to interferences and measurement noise. Oppositely, in a bounded error context, measures and modelling errors are supposed to be unknown but to stay within known and acceptable bounds.

Errors between measured and predicted outputs may rely on many factors, among them: limited sensors accuracy, interferences, noise, structured uncertainties. . . Some are quantifiable, some are not. We consider here the quantifiable error $e$, which is added to the model output $y$. The experimental outputs $y_{\text{exp}}$ are given by:

$$y_{\text{exp}}(t_j, p) = y(t_j, p) + e(t_j, p), \quad 1 \leq j \leq n. \quad (3)$$

In the presented work, the error $e$ is supposed to be within an interval whose lower bound is $e_{\text{min}}$ and upper bound is $e_{\text{max}}$. An allowable error set $\mathbb{E}$ may be defined as a set of constraints

$$\mathbb{E} = \{ e(t_j, p) \mid e_{\text{min}} \leq e(t_j, p) \leq e_{\text{max}} \}. \quad (4)$$

These bounds may be considered constant over time as well as variable. They may be established from data given by constructors for electronic parts for example.

Our system has unknown but bounded initial conditions while input and output values are available at any time. The initial conditions belong to a set, hence the model output $y$ is also a set denoted $[y]$, as well as the error $e$ which is a set $[e]$ that must be in the domain $\mathbb{E}$.

![Figure 1: System and model.](image)

In the same way as for $[e]$, we define an allowable domain $\mathbb{Y}$ for model output $[y]$ such that

$$\mathbb{Y} = \{ y \mid y \subseteq [y_{\text{exp}} - e_{\text{max}}, y_{\text{exp}} - e_{\text{min}}] \}. \quad (5)$$

Interval analysis is used to reject models that are not consistent with data and error bounds.

Numerous approaches have been tested with linear models: ellipsoid shaped methods [Milanese and Vicino, 1991; Durieu and Walter, 2001; Leseqc et al., 2003], parallelotopic and zonotopes [Alamo et al., 2005].

Consider a nonlinear parametric model described by the following set of equations:

$$\begin{cases} \dot{x}(t, p) = f(x(t), u(t), p), \\ y(t, p) = g(x(t), u(t), p), \\ x_0 \in X_0, \\ p \in P_0, \end{cases} \quad (6)$$

where

- $f$ and $g$ are continuous nonlinear known functions,
- $x(t) \in \mathbb{R}^n$ is the state vector at time $t$,
- $u(t) \in \mathbb{R}^m$ is the input vector at time $t$,
- $y(t) \in \mathbb{R}^p$ is the output vector at time $t$,
- $X_0$ is an a priori known set enclosing the initial condition $x_0$,
- $P_0$ is an a priori known set enclosing the searched parameter vector $p$.

A parameter vector $p$ is acceptable if and only if the error between $y_{\text{exp}}$ and the model output $[y]$ is bounded in a known way. To estimate system parameters, we have to get the set $\mathbb{P}$ of all parameters $p$ enclosed in the a priori search set $P_0$ such that error between real data and model outputs denoted

$$[e(t, p)] = y_{\text{exp}} - [y(t, p)]$$

belongs to the allowable error set $\mathbb{E}$ whose bounds $e_{\text{min}}$ and $e_{\text{max}}$ are known:

$$\mathbb{P} = \{ p \in P_0 \mid [e(t, p)] \subseteq \mathbb{E} \}, \quad (8)$$

The characterization of the set $\mathbb{P}$ may be defined as a set inversion problem [Raïssi et al., 2003; Kieffer and Walter, 2005]:

$$\mathbb{P} = [e^{-1}](\mathbb{E}). \quad (9)$$

A guaranteed approximation of $\mathbb{P}$ may be computed using the SIVIA algorithm presented previously.

4 Case study

4.1 Problem

One of the tasks devoted to flight control computer is to slave the position of the control surfaces. The control surface motion is driven by an actuator in active or damped mode. There are generally two actuators for one control surface. A master computer performs control by sending a command on the active actuator. The other one is set in damped mode and follows the surface motion. When the master computer detects a failure, it switches the active actuator to damped mode and gives control to a slave computer that controls the second actuator which is now in active mode.

Some parts in the control chain that contain electronic devices may generate spurious oscillating signals. Under some circumstances, these signals can lead to an unwanted oscillation of the control surface. This is called an Oscillatory Failure Case (OFC). In this paper, only OFC located in the servo-loop control of the moving surfaces are considered, that is, between the Flight Control Computer and the control surface, including these two elements (cf. Figure 2). When an OFC occurs within the actuator bandwidth, it may have the following consequences:

- It creates additional loads on the aircraft structure. Coupled with the aeroelastic behaviour of the aircraft, it may lead to unacceptably high loads or vibrations (resonance phenomena with aircraft natural modes).
- It speeds up actuators stress and reduces their lifetime.
Figure 2: Position control chain.
• It lowers passengers comfort.

The plane is designed to take into account these faults in a limited way, depending on oscillation frequency and range. Taking structural design actions to counteract these faults would indeed require heavily and costly structure reinforcement. It is then very much advisable to detect them using the flight control computers. Monitoring must be performed to ensure that failures stay within predefined limits. Classical monitoring (e.g. position monitoring, runway monitoring, etc.) does not guaranty such detections, so specific mechanisms must be added.

When an OFC is detected, the flight computer looses regulation over control surface. As seen previously, another waiting computer ensures surface control with a redundant servo which switches from damped to active mode.

The problem to be solved is to detect in the control loop some OFC with a minimal given range within a given number of periods (the maximal overload does not immediately occur on the structure but after some periods of oscillation). For example, it could be required to detect a 1° failure within 3 periods, on a given control surface, between 0.2 and 5 Hz. This goal has been chosen for this paper. In real cases, it depends on the a/c type and control surfaces.

4.2 Liquid vs. solid failures
Two different kinds of OFC may occur: liquid or solid ones. As shown in the scheme of figure 3, a liquid failure is a erroneous signal added to the control loop signal. A solid failure is a signal which replaces the control loop one.

In both cases, a failure is a periodic sinus shaped signal whose frequency, range and phase obey to an uniform law. For both cases of failure, residuals corresponding to estimated position subtracted from real position are shown in Figure 4.

These residuals are used to detect the OFC. The current method used in A380 flight control computers relies on residual evaluation by oscillation counting inside spectral subband [Goupil, 2007]. This computation use the same kind of model as the one shown on figure 5. Its complexity can be increased depending on the kind of failures we want to detect on a given control surface.
Figure 4: Residuals: liquid failure case on the left side, solid failure case on the right side.

5 Application

In the following section, we address the case of liquid failure with the bounded error parameter estimation method presented previously and use this estimation for detecting OFC. The results are analyzed with respect to the currently used detection method.

Our goal is to perform parameter estimation of the liquid failure model. This fault model defines the shape of the position signal as either a sinus or a triangle. The system to monitor is a simple model of a control surface whose motion is ensured by a hydraulic servo command as presented in figure 5.

In this model, \( o(n) \) is the position control signal at time \( n \). The control error \( \varepsilon(n) \) is given by:

\[
\varepsilon(n) = o(n) - \hat{s}(n-1). \tag{10}
\]

It is the difference between the position control \( o \) at time \( n \) and the estimated position \( \hat{s} \) at time \( n - 1 \). The estimated current \( \hat{i}(n) \) is proportional to the error:

\[
\hat{i}(n) = K \varepsilon(n) \tag{11}
\]

where \( K \) is the constant control gain. The current is then converted to speed \( \hat{v}(n) \). Finally, the estimated control surface position \( \hat{s}(n) \) at time \( n \) is computed by integration of the speed.

We ran tests introducing oscillatory failures in the control loop. Two fault models, triangle shaped and sinus shaped, were used. Parameters were estimated over one period of the signal.

5.1 Sinus shaped fault

A high noisy sinus-shaped liquid fault signal with a range \( A = 1^\circ \) and a frequency of \( f = 0.5 \)Hz is introduced in the control surface model. The initial parameter box is given by \( A \times f = [0, 3] \times [0, 10] \).

Figure 6 shows the results provided by the set inversion algorithm when the fault model is supposed to be sinus-shaped. Range parameter \( A \) is showed on the horizontal axis while frequency \( f \) is on the vertical one. Blue boxes have been rejected, yellow ones have a length inferior to the stop condition set in the algorithm. The red boxes represent the solution. We notice that they concentrate around the real parameter values.

When the fault model is triangle-shaped, the algorithm stops after a few iterations and its conclusion is the non-existence of a solution.

5.2 Triangle-shaped fault

In this example, the fault is triangle-shaped with a range \( 2^\circ \) and a frequency of \( f = 0.5 \)Hz, with a still highly noisy signal. The initial parameter box is now \( A \times T = [0, 3] \times [0, 5] \), with \( T = 1/f \).

Figure 7 exhibits the obtained results with a triangle-shaped fault model. The parameter \( A \) is on the horizontal axis and the period \( T \) on the vertical axis. One can notice that
the estimation results are fully in accordance with the injected fault.

Another direction to go is to use alternate set membership detection methods under the condition to have proper surface control loop models. State estimation and parity state methods, both using interval analysis, should be tested.

6 Discussion and conclusion

In this paper we presented a method for failure detection using fault models and an error bounded estimation method. The method is based on interval analysis which provides guaranteed results in an error bounded context. It has been tested to detect plane control surfaces oscillatory failures.

The tests show good results for confirming a fault. Now, the real advantage of the method with respect to others is that it is very efficient to prove the non-existence of a solution.

Future work will consist in improving the fault model detection by studying its properties: response time, false alarm rate, non detection rate and robustness. More simulation tests using alternate fault models against real data will also be performed.

References


Another Point of View on Diagnosability

Xavier Pucel, Louise Travé-Massuyès, Yannick Pencolé
LAAS-CNRS ; Université de Toulouse ; 7, avenue du Colonel Roche, F-31077 Toulouse, France
{xpucel, louise, ypencole}@laas.fr

Abstract

This paper provides a new definition of diagnosability, that allows one to check the diagnosability of any set of system states, by extension of properties that depend on the system state. The existing definitions and approaches for checking diagnosability apply to faults or sets of faults, and comparison shows that the new definition generalizes the existing ones. This new definition is applied to repair preconditions, and an example shows how this brings complementary information compared to classical fault diagnosability.

1 Introduction

Complex and critical systems require close supervision when running, and the model-based community has produced a lot of work in this area. In particular, model-based diagnosis is an increasingly active research domain, and the problem of diagnosability analysis has been addressed many times [Sam-path et al., 1995; Cordier et al., 2006; Bayoudh et al., 2006]. In a diagnosable system, although it is impossible to know the exact state of the system, the supervisor is aware of which anticipated faults have happened and which have not. However, the information needed by a supervisor in such systems is not limited to fault presence, and fault diagnosability in a system does not guarantee that this system is easy to supervise. A system designer hence needs to verify the diagnosability of more than just the faults.

This paper presents a definition for diagnosability that can be applied to any set of states in a system. The model used in this paper is state-based, and most properties can be mapped to a set of system states. This provides the possibility to check the diagnosability of state dependent properties. When a property is diagnosable, the system supervisor is always able to assess whether the current system state verifies this property. Fault presence or absence are examples of such state dependent properties. The definition can also be used to perform diagnosability analysis for faults, in the same way as existing approaches do. But its extension to any set of states also allows to check the diagnosability of many other properties, like for example repair preconditions, which is illustrated in this paper.

The existing diagnosability approaches are presented in first place, before the new definition is introduced. This new definition is compared to existing definitions. Finally, an example illustrates the application of the new definition to repair preconditions.

2 Related work

Diagnosis has been an active research topic in AI for many years and numerous approaches have been proposed to cope with on-line as well as post mortem diagnosis [Hamscher et al., 1992]. More recently a significant trend has moved the activities of the diagnosis community towards the analysis of the properties related to diagnosis. Several pieces of work deal with defining and checking diagnosability [Console et al., 2000; Travé-Massuyès et al., 2001; Travé-Massuyès et al., 2006; Struss and Dressler, 2003; Bayoudh et al., 2006] and a unified characterization bridging state based and event based modeled systems has been proposed [Cordier et al., 2006]. Diagnosability guarantees that all the anticipated faulty situations of a system are discriminable one from the other, although the state of the system is partially observed. This property is quite important because it indicates that the instrumentation providing the observations about the system is well designed and sufficient to provide an explanation of what is going on. However, nowadays systems are required to run more and more autonomously and they are expected to cope with unanticipated situations by themselves, in particular when faults occur. Hence, diagnosability has been more recently addressed together with the requirements for repairability in order to provide a formal definition for self-healability [Cordier et al., 2007].

3 Existing diagnosability approaches

This section presents diagnosability approaches existing in the literature, in order to compare them to the new approach defined in section 4. All approaches rely on a formal description of the system behaviour, in the absence and in the presence of faults.

In existing approaches, diagnosability is defined as the ability of a system to exhibit different observations for a predefined set of faults. It is based on the notion of signature, which maps faults to sets of observations.
3.1 System representation

The system is assumed to be described by a proposition $sd$ which can be expressed in propositional logic. The set of models of this proposition is denoted $SD$, it contains all variable tuples satisfying $sd$ and describes the set of all the system states, faulty or non faulty. The set of variables is denoted $V$. Some of the variables characterize the presence or absence of faults, these are called mode variables, $O$ denotes the set of observable variables. Generally, mode variables are not observable. The set $OBS$ contains all the possible system observations. In other words, it contains the models of the restriction of $sd$ to the variables in $O$.

3.2 Faults and fault modes

Various faults may occur in the system, modifying its behaviour and possibly making it unable to fulfill its function. Several faults may be present at the same time.

A fault mode characterizes the behaviour of the system under a given combination of faults. It assesses the presence of some faults as well as the absence of the other faults. The normal mode is one of the many fault modes, it assesses the absence of all faults. The occurrence of a permanent fault changes the fault mode of the system.

A fault is characterized by one mode variable, whose value indicates whether the fault is present or not. A fault mode is identified by a value for the tuple of all mode variables. $SD_f$ is the description of the fault mode $f$, i.e. the set of states in which the fault mode is verified. As any state belongs to exactly one fault mode, the set of all $SD_f$ is a partition of $SD$, as illustrated in Figure 1.

3.3 Projection on observable variables

An operation called projection on observable variables and noted $\mathcal{P}_{obs}$ is used. It takes as input a system state expressed as a variable value tuple, and outputs the tuple of observable variable values obtained in this state. For example, if $V$ contains 5 variables, and if the first and third are observable, then:

$$\mathcal{P}_{obs}(v_1, v_2, v_3, v_4, v_5) \rightarrow (v_1, v_3)$$

The inverse projection $\mathcal{P}^{-1}_{obs}$ is defined from $OBS$ to $2^{SD}$ as follows:

$$\mathcal{P}^{-1}_{obs}(\sigma) = \{ s \in SD, \mathcal{P}_{obs}(s) = \sigma \}$$

The projection $\mathcal{P}_{obs}$ associates a system state to the observation that is received under this state. The inverse projection associates an observation to the set of all states that may have originated this observation. When applied to fault modes, this projection is the base of the classical diagnosability analysis approaches.

3.4 Fault mode diagnosability

The classical definition for fault signature and diagnosability is provided now. The definition of diagnosability states that the system cannot produce a common observation under two different fault modes [Cordier et al., 2006].

Definition 1 (Fault mode Signature and Diagnosability)

The signature of a fault mode $f$ is the set of all possible observations when the system state belongs to the mode $f$.

$$Sig(f) = \{ \mathcal{P}_{obs}(s), s \in SD_f \}$$

A system is diagnosable if and only if, $f_1$ and $f_2$ being fault modes:

$$\forall f_1, f_2 \quad f_1 \neq f_2 \Rightarrow Sig(f_1) \cap Sig(f_2) = \emptyset$$

When diagnosability according to definition 1 holds, the observations emitted by the system always allow one to decide which faults have happened, and which faults have not. But when the signatures of two fault modes intersect, this means that there exists at least one observation that can be emitted by the system under two different fault modes. There are two possible explanations for this observation, and a diagnosis process would output two diagnostic candidates.

3.5 Macrofault diagnosability

Another definition of diagnosability is given in [Cordier et al., 2007] as an extension of definition 1. It is based on the notion of macrofault, which is a set of fault modes. It is based on the idea that not all pairs of fault modes need to be discriminable: fault modes that do not need to be discriminated one from another are gathered into a macrofault. The set of states in which the macrofault $F_i$ is present is noted $SD_{F_i} = \bigcup_{f \in F_i} SD_f$.

This raises a significant difference compared to the previous approach. Whereas fault modes are disjoint, macrofaults may overlap. In the macrofault approach, it is considered that when the system state belongs to several macrofaults, it belongs to an overlapping fault mode, and identifying only one of the macrofaults with certainty is enough for the system to be diagnosable.

In this approach, only covering sets of macrofaults are considered, i.e. sets of macrofaults such that every fault mode belongs to a macrofault. Consequently, there is always at least one present macrofault, whatever the system state is.

Definition 2 (Macrofault, Characteristic signature)

A macrofault $F_i$ is a set of fault modes. $F_i$ is present if and only if the system is in one of the fault modes $f \in F_i$.

A characteristic signature $cSig(F_i)$ is a set of observations that allow one to assess with certainty that the macrofault $F_i$ is present.

$$cSig(F_i) \subseteq \left( \bigcup_{f \in F_i} (Sig(f_i)) \right) \setminus \bigcup_{f_k \notin F_i} (Sig(f_k))$$

Note that there are several possible characteristic signatures for each macrofault. If $O$ is a characteristic signature for a macrofault $F_i$, then any $O' \subseteq O$ is also a characteristic signature for $F_i$.

Definition 3 (Macrofault Diagnosability)

A covering set of macrofaults $\{F_i\}$, i.e. a set of macrofaults that cover all the fault modes, is diagnosable if and only if there exists a set of characteristic signatures for these macrofaults that form a partition of $OBS$. 

When such a partition is established as illustrated in Figure 1, it is always possible to find out at least one present macrofault. As a state may belong to several macrofaults, an observation can also correspond to several macrofaults. However, it is only needed, for each observation, to assess with certainty that one macrofault is present.

This definition is a generalization of definition 1, since fault modes are particular macrofaults. Because macrofaults may overlap when they contain the same fault mode, this definition applies to a greater range of sets of states than the fault mode definition.

This definition is also less constrained than fault mode diagnosability (definition 1), in the sense that in a system verifying fault mode diagnosability, any set of macrofaults is diagnosable.

4 Diagnosability revisited

This section presents a new definition of diagnosability, which applies to any state dependent property. It is based upon the analysis of the set of states in which a property holds. It is a generalization of existing diagnosability definitions which only apply to sets of states characterized by the presence or absence of some faults. Comparisons show that this new definition is consistent with the existing ones.

4.1 Diagnosability of a property

Definition 4 (Diagnosable block) Let $=_{\text{obs}}$ be the equivalence relation defined on $SD$ by:

$$\forall s_1, s_2 \in SD, s_1 =_{\text{obs}} s_2 \iff \mathcal{P}_{\text{obs}}(s_1) = \mathcal{P}_{\text{obs}}(s_2)$$

Each equivalence class of $=_{\text{obs}}$ is called a diagnosable block of the system. The set of diagnosable blocks of the system is the quotient set of $SD$ by $=_{\text{obs}}$.

Definition 5 (Diagnosability) A property or its corresponding set of states $S \subseteq SD$ is diagnosable if and only if $S$ is exactly a union of diagnosable blocks.

Figure 2 depicts a system with 7 states and 4 possible observations. The diagnosable blocks are represented by white sets with dashed lines. Observation $o_2$ is received in two different states, one inside $S_1$ and one outside. Thus, when observing $o_2$, a supervisor is unable to decide whether the system is in $S_1$ or not. On the other hand, it is always possible to decide from the observations whether the system state belongs to $S_2$ or not.

4.2 Comparison with fault mode diagnosability

Since definition 5 applies to any set of states, it applies in particular to fault modes. It is shown now that when applied to fault modes, this definition is equivalent to definition 1.

Proposition 1 A system is diagnosable according to definition 1 if and only if for every fault mode $f$, $SD_f$ is diagnosable according to definition 5.

Proof

The signatures of two fault modes $f_i$ and $f_j$ intersect if and only if there exists a state $s_i \in SD_{f_i}$ and another state $s_j \in SD_{f_j}$ leading to the same observation. These two states obviously belong to the same diagnosable block, say $d$, and, since $SD_{f_i}$ and $SD_{f_j}$ are disjoint, none is a superset of $d$. Since diagnosable blocks form a partition of $SD$, $s_i$ (resp. $s_j$) does not belong to any other diagnosable block than $d$. Hence, $SD_{f_i}$ (resp. $SD_{f_j}$) is not a union of diagnosable blocks.

If one fault mode $f_i$ is not a union of diagnosable blocks, then since fault modes form a partition of $SD$, there exists a diagnosable block $d$ containing a state $s_i$ of $f_i$ and at least one state $s_j$ belonging to another fault mode $f_j$. These two states lead to the same observation $o$, which necessarily belongs to both $\text{Sig}(f_i)$ and $\text{Sig}(f_j)$. Consequently the signatures of all fault modes are not disjoint.

4.3 Signature and preemptability

Definition 5 expresses the diagnosability of a single property. This definition is now extended to a set of properties. For this, the classical notion of signature is extended and the notion of preemptability is introduced. The new definition of the signature applies to sets of states as opposed to definition 1 that applies to fault modes.

Definition 6 (Signature of a set of states) The signature of a set of states $S$, or of the property $p$ mapped to $S$, is the set of observations that can be obtained when the system is under one of these states:

$$\text{Sig}(S) = \{\mathcal{P}_{\text{obs}}(s), s \in S\}$$
This definition applies equally to the complement set $S$. As sets of states generally overlap, comparing their signatures with one another does not bring much information. It is worth-while to compare their signatures with the signatures of their respective complements. Indeed, if a set of states corresponds to a given property of the system, its complement corresponds to the negation of the property.

**Definition 7 (Diagnosable space, Undiagnosable space)**

The diagnosable space $D(S)$ (resp. undiagnosable space $UD(S)$) of a set of states $S$ mapped to a property $p$ is the subset of $S$ in which it is possible (resp. impossible) to assert whether the property $p$ holds.

$$UD(S) = S \cap \mathcal{P}_{OBS}^{-1}(Sig(S))$$

$$D(S) = S \setminus UD(S)$$

As illustrated in Figure 3, $D(S)$ can also be defined as the union of the diagnosable blocks included in $S$. The diagnosable blocks that intersect but are not included in $S$ form $UD(S) \cup UD(S)$. The intersection of this set with $S$ gives $UD(S)$. Hence, when a set of states is diagnosable, its undiagnosable space is empty.

When a property $p$ is undiagnosable, it can be preemptable if its undiagnosable space is included in the diagnosable space of other properties. In this case these other properties may preempt $p$ in the sense that when the validity of $p$ is uncertain, one of these other properties is valid, which makes $p$ unnecessary.

**Definition 8 (Preemptability)** A property or its corresponding set of states $S$ is preemptable if and only if:

$$UD(S) \subseteq \bigcup_{S' \neq S} (D(S'))$$

Figure 4 illustrates a set $S_0$, whose undiagnosable space is included into two diagnosable sets $S_1$ and $S_2$.

### 4.4 Diagnosability of a set of properties

This section presents a definition of diagnosability for a set of properties that accounts for the mutual influence that properties may have with one another by the means of preemptability.

**Definition 9 (Diagnosability of a set of properties)** A set of properties is diagnosable if and only if each property is either diagnosable or preemptable.

Considering a diagnosable set of properties, the union of all the corresponding sets of states is diagnosable.

Let $S_i$ be the set of states corresponding to the $i$-th property of a diagnosable set of properties. For each $i$, $UD(S_i)$ is either empty or included in the diagnosable sets of other sets of states. Hence, $\bigcup S_i = \bigcup D(S_i)$ is diagnosable since each $D(S_i)$ is a union of diagnosable blocks.

### 4.5 Comparison with macrofault diagnosability

Now it is shown that definition 9 is equivalent to definition 3 when applied to macrofaults.

**Proposition 2** A covering set of macrofaults is diagnosable according to definition 3 if and only if it is diagnosable according to definition 9.

**Proof**

First, given a macrofault $F_i$, let us consider $Sig(D(F_i))$. This set contains no observation from a state in which $F_i$ is absent, and is hence a characteristic signature for $F_i$. Let us map each macrofault $F_i$ to the set $\Sigma_i = Sig(D(F_i)) \setminus \bigcup_{j < i} Sig(D(F_j))$. $\Sigma_i$ is a characteristic signature for $F_i$, since it is a subset of $Sig(D(F_i))$.

Second, let $o \in Sig(D(F_i))$. We have either $o \notin \Sigma_i$, or $o \in \Sigma_j$ with $j < i$, and if $o \in \Sigma_i$ then necessarily $o \notin \Sigma_k$ with $k \neq i$. Hence, the set of all $\Sigma_i$ forms a partition of the set $\bigcup Sig(D(F_i))$.

The previous statements are now used to establish the equivalence. The covering set of macrofaults $\{F_0 \ldots F_n\}$ is diagnosable according to definition 9 if and only if the set of all $D(F_i)$ covers $SD$ (see section 4.4), which is equivalent to the set $\bigcup Sig(D(F_i))$ covering $OBS$. Consequently, from the statements above, it follows that the set $\Pi = \{\Sigma_0 \ldots \Sigma_n\}$ partitions $OBS$, and the set of macrofaults is diagnosable according to definition 3.

### 5 Application to repair preconditions

The definition of diagnosability is now applied to sets of states that map to repair preconditions. When a repair precondition is diagnosable, it is possible to decide when to apply the repair and when not. It is a complementary approach to fault diagnosis, since knowing which fault happened and knowing what to do to repair it is not the same information, and both answers are important for monitoring a system.
Knowing which fault happened but being unable to decide which repair is suited is odd. On the other hand, knowing how to repair a faulty system without knowing the details of the faults is a problem for low cost maintenance or feedback to the system designers. Hence, diagnosability analysis of repair preconditions is a complement to fault diagnosability analysis.

A repair is an action or a process that puts a system back to a normal state from a faulty abnormal state. Repairs can be plans driven by goals [Friedrich et al., 2005], reconfigurations [Ten Teije et al., 2004], or other actions. In most approaches, repairs have preconditions, which generally define a set of states. In the case of repair plans, the plan may contain actions that bring additional information about the system state, thus refining the diagnosis. Plans may also contain conditional branchings, especially in order to react to additional diagnosis information.

A repair may not be executed in every state of the system for various reasons. An action or plan that repairs a system from a given state may damage it even more in some other states. For example changing a wheel is not possible if the vehicle is not at full stop. Also, it is considered in this paper that repairs being the system back in a normal state, partial repairs are not considered. For example, changing one wheel repairs a vehicle with one flat tire, but it does not repair a vehicle with two flat tires.

In most cases, non faulty states do not belong to repair preconditions, since it is not useful to repair a normal system. However, when there is an ambiguity about the presence of a fault, some supervision policies consider that it is better to repair a normal system is better than let the system run with its fault. Consequently, normal states may belong to fault preconditions.

**Definition 10 (Repair precondition)** A repair precondition is a set of states, in which the repair can be applied, and in which the application of the repair brings the system to a normal state.

This definition implies that if two repair preconditions are verified at the same time, only one repair needs to be applied.

No assumption is made in this paper about the relation between fault modes and repair preconditions. A repair may be applicable in only some of the states of a fault mode, while each fault mode may be repaired differently according to the current system state.

Each repair precondition is described by a logic proposition $r_{p_i}$. The proposition $r_{p_i}$ generally constrains mode variables as well as variables defining the operational state of the system. The set $\mathcal{RP}_i \subseteq \mathcal{SD}$ contains all the system states fulfilling $r_{p_i}$. The set $\overline{\mathcal{RP}_i}$ is the complement of $\mathcal{RP}_i$ in $\mathcal{SD}$, it is the set of system states for which the $i$-th repair is not suited.

6 Example

The concepts and definitions described in the previous sections are illustrated by a simple example. It is shown that definitions 5 and 9 allow us to analyse diagnosability at different levels (faults, macrofaults, or repair preconditions) and that the returned information may be different and complementary.

**System** The system consists in a fluid pipe with variable input flow, which is supposed to provide a constant output flow. A tank is used to compensate flow variations. This tank is filled when the input flow is higher that the expected output, and provides water when the input flow is too low.

**Faults** Two faults are considered. First, the pipe may be clogged, which reduces greatly the flow capacity of the pipe. Second, the tank is supposed to be always able to deliver water, however in exceptional conditions, the tank may occur to be empty. When this occurs, the input flow is directed in priority to the tank. If the input flow is sufficiently high, it can supply both the empty tank and the output.

**Sensors** A pressure sensor is placed in the pipe, in order to detect abnormally high pressures. This happens when the pipe is clogged, and there is input flow.

**Model** The model of this system contains five variables:

- $if$: describes the input flow and has 3 values: none, low, and high.
- $of$: is Boolean and equals 1 when there is an output flow.
- $hp$: is Boolean and equals 1 when there is an output flow.
- $pc$: is Boolean and equals 1 when the pressure inside the pipe is abnormally high.
- $te$: is Boolean and equals 1 when the tank is empty.

The behaviour of the system is described by the following constraints:

$$of = 1 \Leftrightarrow (te = 0 \lor (if = high \land pc = 0))$$

$$hp = 1 \Leftrightarrow (if \neq none \land pc = 1)$$

**Observables** The variables $if$, $of$, and $hp$ are observable.

**Repairs** The following repairs are available:

1. It is possible to unclog the pipe thanks to a chemical action ($rp_1$). This repair can be applied when the pipe is clogged. For safety reasons, it must not be applied when the pipe is not clogged. If the tank is empty, this repair is not sufficient to bring back the system in a normal state.

$$rp_1 : (pc = 1 \land te = 0)$$
2. It is also possible to unclog the pipe mechanically (rp2). The action consists in sending someone on site and clean the pipe. This repair can only be applied when the pipe is empty (no input flow), but is not sufficient if the tank is empty, since it will not bring the system to a normal state. This repair can if necessary be applied in the normal mode: once the cleaner is on site, if the pipe is not clogged, then the cleaner will do nothing.

\[ rp_2 : (te = 0 \land if = \text{none}) \]

3. Finally, if the tank is empty, it is possible to redirect the whole flow through the tank (rp3). This permits to mechanically unclog the pipe if needed. However, the manipulations involved in this repair require that there is no flow in the pipe.

\[ rp_3 : (te = 1 \land (if = \text{none} \lor pc = 1)) \]

The system has 12 different states, represented in Figure 6 as well as the diagnosable blocks and their corresponding observations. The application of definitions 5 and 9 to the fault modes, macrofaults and repair preconditions are now illustrated on this system.

6.1 Fault mode diagnosability analysis

The system has 4 fault modes, named normal, pipe clogged, tank empty and pipe & tank that define 4 sets of states whose diagnosability is analyzed. The analysis details are described in Table 1.

According to definition 5, none of these fault modes is diagnosable. Moreover, fault modes are by definition disjoint sets, and the notion of preemptability is not relevant when dealing with disjoint sets of states, since disjoint sets cannot preempt one another. Consequently, no fault mode is diagnosable according to definition 9 either.

6.2 Macrofault diagnosability analysis

Let us consider for example the set of macrofaults defined by \{F_1, F_2, F_3\} with \[ F_1 = \{\text{normal, tank empty}\} \text{ and } F_2 = \{\text{pipe clogged}\} \text{ and } F_3 = \{\text{tank empty, pipe & tank}\}. \]

The diagnosability analysis is given in Table 2.

None of these macrofaults is diagnosable with respect to definition 5. Moreover, only \[ F_3 \] is preemtable, with \[ UD(F_3) \subseteq D(F_1) \] which is not enough to make the set of macrofaults \{F_1, F_2, F_3\} diagnosable according to definition 9.

6.3 Repair precondition diagnosability analysis

The sets of states corresponding to the repair preconditions are represented in figure 6. Diagnosability analysis provides the results indicated in Table 3.

The undiagnosable spaces of repair preconditions \( \mathcal{RP}_p \) and \( \mathcal{RP}_2 \) are empty, which means these sets of states are unions of diagnosable blocks. They are diagnosable, according to definition 5. Moreover, \( \mathcal{RP}_1 \) is not diagnosable, but \[ UD(\mathcal{RP}_1) \subseteq D(\mathcal{RP}_1) \], it is preemptable. The set of repair preconditions \( \{\mathcal{RP}_1, \mathcal{RP}_2, \mathcal{RP}_3\} \) is diagnosable with respect to definition 9.

7 Conclusion

This paper provides a new, general definition of diagnosability, that applies to any set of states and by extension to any state dependent property. Given a property that depends on the system state, it is possible to assess whether the observations are sufficient to deduce that the property holds or not. Such properties can be the presence or absence of faults, which falls back into existing diagnosability approaches. They can also be repair preconditions, which we expect to fall back into existing self-healability approaches [Cordier et al., 2007]. Analysing repair preconditions diagnosability is showed to bring different information from analysing fault modes or macrofaults diagnosability.

This work covers many other kinds of properties, like for example the ability of the system to provide its function, or a given part of its function, or to respect a given quality of service. Any property that can be mapped to a set of states can be checked with this approach.

References


[ten Teije et al., 2004] A. ten Teije, F. van Harmelen, and B. Wielinga. Configuration of web services as parametric
Figure 6: States, diagnosable blocks and repair plans of the system.

### Table 1: Fault modes diagnosability results.

<table>
<thead>
<tr>
<th>Fault mode</th>
<th>States</th>
<th>Intersected diagnosable blocks</th>
<th>( UD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_D^{\text{normal}} )</td>
<td>( {s_1, s_5, s_9} )</td>
<td>( {s_0, s_1} ) and ( {s_9, s_{10}} )</td>
<td>( {s_1, s_9} )</td>
</tr>
<tr>
<td>( S_D^{\text{pipe clogged}} )</td>
<td>( {s_0, s_4, s_8} )</td>
<td>( {s_0, s_1} )</td>
<td>( {s_0} )</td>
</tr>
<tr>
<td>( S_D^{\text{tank empty}} )</td>
<td>( {s_2, s_6, s_{10}} )</td>
<td>( {s_9, s_{10}} ) and ( {s_2, s_3} )</td>
<td>( {s_2, s_{10}} )</td>
</tr>
<tr>
<td>( S_D^{\text{pipe &amp; tank}} )</td>
<td>( {s_3, s_7, s_{11}} )</td>
<td>( {s_2, s_3} )</td>
<td>( {s_3} )</td>
</tr>
</tbody>
</table>

### Table 2: Macrofaults diagnosability results.

<table>
<thead>
<tr>
<th>Macrofault</th>
<th>States</th>
<th>Intersected diagnosable blocks</th>
<th>( UD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1 )</td>
<td>( {s_1, s_2, s_5, s_6, s_9, s_{10}} )</td>
<td>( {s_0, s_1} ) and ( {s_2, s_3} )</td>
<td>( {s_1, s_2} )</td>
</tr>
<tr>
<td>( F_2 )</td>
<td>( {s_0, s_4, s_8} )</td>
<td>( {s_0, s_1} )</td>
<td>( {s_0} )</td>
</tr>
<tr>
<td>( F_3 )</td>
<td>( {s_2, s_3, s_6, s_7, s_{10}, s_{11}} )</td>
<td>( {s_9, s_{10}} )</td>
<td>( {s_{10}} )</td>
</tr>
</tbody>
</table>

### Table 3: Repair preconditions diagnosability results.

<table>
<thead>
<tr>
<th>Repair precondition</th>
<th>States</th>
<th>Intersected diagnosable blocks</th>
<th>( UD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_P^1 )</td>
<td>( {s_0, s_4, s_8} )</td>
<td>( {s_0, s_1} )</td>
<td>( {s_0} )</td>
</tr>
<tr>
<td>( R_P^2 )</td>
<td>( {s_0, s_1} )</td>
<td>none</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( R_P^3 )</td>
<td>( {s_2, s_3, s_7, s_{11}} )</td>
<td>none</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

In each table, the column “Intersected diagnosable blocks” lists the diagnosable blocks that intersect but are not subset of the corresponding set of states.


Combining State Estimation and Simulation in Consistency-based Diagnosis using Possible Conflicts

Belarmino Pulido and Aníbal Bregón and Carlos Alonso-González
GSI, Departamento de Informática, Universidad de Valladolid, E-47014. Valladolid, Spain
email: {belar,anibal,calonso}@infor.uva.es

Abstract
Consistency-based diagnosis of dynamic systems using possible conflicts relies upon semi-closed loop simulation of numerical models. Simulation approaches need to know the initial state. This assumption is not easily fulfilled in real systems, even in the presence of measurements related to state variables due to noise and parameter uncertainties.

This work proposes to integrate state observers to estimate initial states for simulation within the consistency-based diagnosis framework using possible conflicts. This work extends the BRIDGE framework for a specific class of dynamic systems, using the possible conflict concept to find every subsystem with the necessary structural redundancy to lead to a minimal conflict activation. Algorithms used to find possible conflicts provide the structure of executable models which can be implemented as simulation models. Without additional information, these algorithms can detect structural models which could be implemented as state observers. These observers can provide the needed estimation of initial values for simulation, and can be used at the same time to speed up the fault detection step in consistency-based diagnosis. Our integration proposal is open to different kinds of state observers—except for its structural model—, and different fault detection configurations.

The proposal has been tested on a simulation scenario. Results and comparison with similar existing hybrid -DX + FDI- approaches are provided.

Keywords: Consistency-based diagnosis, State-observers,

1 Introduction
The Fault Detection and Isolation community (FDI) born in the Automatic Control world, and the Diagnosis community (DX) born in the Artificial Intelligence field, have approached in parallel Model-Based Diagnosis (MBD) using different but complementary ways.

FDI community uses control and statistical decision theories to carry on the fault detection and isolation stages. Concerning fault detection, a set of fault indicators, called residuals, are generated in such way that they become active in presence of some faults, but non active in presence of others. Residual generation techniques can be divided into two main categories: state estimation, and parameter identification [Gertler, 1998].

In the FDI community, there are solid theoretical results for linear systems [Gertler, 1998; Patton et al., 2000; Blanke et al., 2003], while analysis of non-linear systems is a major research issue. A major topic in FDI research is robustness in fault detection stage, which mainly uses numerical models and must take into account 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 et al., 2000].

The DX community is rooted in the Artificial Intelligence field. The main issues in DX community are the fault localization and identification stages. The theoretical background of this approach is solid for static systems. It’s main theoretical framework is consistency-based diagnosis (CBD), and GDE is its computational paradigm [Hamacher et al., 1992].

Both DX and FDI communities have developed their own tools and techniques. Recently, the BRIDGE community [Cordier et al., 2004] established a common framework for sharing results and techniques. The basis of the work is the comparison between consistency-based diagnosis via conflicts [Hamacher et al., 2002] and fault detection and isolation via analytical redundancy relations (ARRs) obtained through structural analysis. This comparison was carried out for static systems.

This work is based on the Possible Conflict (PC) approach [Pulido and Alonso, 2000], an off-line dependency compilation technique from the DX community, which also fits within the BRIDGE framework. Consistency-based diagnosis using possible conflicts performs on-line behaviour estimation in semi-closed loop simulation—each possible conflict is able to simulate the behaviour of one subsystem—. If no discrepancy is found, initial values for state variables are injected based on available knowledge. The main goal of this work is to improve robustness through a more precise estimation of the initial state for simulation, without modifying its fault isolation capabilities and the consistency-based approach.

Possible Conflicts can be considered as ARR-like struc-
lectures. Recently, it was demonstrated the strong similarities among Possible Conflicts, Conflicts, and ARRs for static systems [Pulido and Alonso-González, 2004]. This work proposes the integration of state observers to estimate the initial state needed for simulation within the consistency-based diagnosis framework using possible conflicts. This work extends the BRIDGE framework for a specific class of dynamic systems, using the possible conflict concept. Algorithms used to find possible conflicts provide the structure of executable models which can be implemented as simulation models. But in some cases, without additional information in the models, the algorithms can detect those structural models which can also be implemented as state observers. These observers will provide the needed estimation of initial values for simulation, and can be used at the same time to speed up the fault detection step in consistency-based diagnosis. In this paper, we propose an integration procedure which is open to different kinds of state observers—except for its structural model—and different fault detection configurations.

The organization of this paper is as follows. First, we summarize basic concepts presented in the introduction such as conflicts, ARRs, and possible conflicts, and we compare the assumptions, techniques, and working principles used for each community to deal with static and dynamic information in the BRIDGE framework. These concepts will be illustrated using a small system which is described afterwards. Based on those concepts, then we remind how the structure of state observers can be derived using possible conflicts. Later on, we propose how to integrate results from state observers with simulated possible conflicts, and results on a simulated system are shown. Finally, discussion with related works and conclusions are provided.

2 Possible conflicts, ARRs, and conflicts in the BRIDGE framework

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.

Computation of PCs is performed on an abstract model for the set of equations in the system description: an hypergraph including just the constrains in the model, and their related known and unknown variables. PCs are obtained off-line via two core concepts: minimal evaluation chains, or MECs, and minimal evaluation models, or MEMs.

MECs are minimal over-constrained sets of relations, and they represent a necessary condition for a conflict to exist. Each constraint in a MEC has one or more variables, and each variable could be solved using the constraint, assuming the remaining variables are known. This fact is called an interpretation for the constraint, i.e. a feasible causal assignment. In the general case, not every interpretation is feasible for non-linear dynamic models.

A MEM is a global consistent causal interpretation for every constraint in a MEC. Using the whole set of available interpretations for each constraint in a MEC, algorithms used to compute PCs are able to find every possible causal interpretation which is globally consistent within a MEC, i.e. the whole set of MEMs for each MEC. Each MEM describes an executable model, which can be used to perform fault detection. Possible Conflicts are defined as the set of relations in a MEC that has, at least, one MEM.

If there is a discrepancy between predictions from these models and current observations, the PC must 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. Further information concerning PC calculation can be found in [Pulido and Alonso, 2000].

Due to space limitations we introduce a brief summary of the similarities and equivalences between PCs, ARRs, and conflicts described in [Pulido and Alonso-González, 2004; Cordier et al., 2004].

Using minimality criterion w.r.t. set of constraints in the model, the whole set of MEMs related to the set of PCs is equivalent to the set of minimal conflicts computed by the GDE.

Moreover, if algorithms used to compute ARRs through structural analysis use such minimality criterion and provide a complete solution—explores every possible causal assignment for every minimal ARR—, the set of PCs has same detection and isolation capabilities as the set of minimal ARRs.

Finally, if every MEM in every PC provides the same solution—what is called the Equivalence assumption in [Pulido and Alonso-González, 2004]—, then PCs, minimal ARRs, and minimal conflicts provide the same solution in terms of fault detection and isolation capabilities.

Cordier et al. [Cordier et al., 2004] introduced the concept of support for an ARR (set of components whose models are used to derive an ARR). Based on such idea, off-line compiled conflicts and ARR’s support can be considered as equivalent (the support for an ARR is a potential conflict, which is equivalent to a possible conflict [Cordier et al., 2004; Pulido and Alonso-González, 2004]). Under given conditions, the set of minimal ARRs and the set of minimal conflicts will have same detection and isolation capabilities.

Summarizing, we can use possible conflict as an equivalent concept to potential conflict, and the support for a minimal ARR. The BRIDGE framework was defined for static systems. This work provides a specific extension for a class of dynamic systems. First, the influence of temporal information in PCs and ARRs calculation must be analyzed. Concepts will be illustrated using the following system.

3 Case study

The system (figure 1) is made up of a water tank, $T$, a valve, $V$, and a PID controller, $C$. The aim of the system is to keep the level of the tank, $h$, which is measured, as close as possible to the desired level, $h_{ref}$. The controller acts through $u_c$ over the valve. Other elements are the input flow sensor, $Q_i$, and the output flow sensor, $Q_o$.

Constraints used to model the system behaviour are:

- **Mass balance**, $c_1$: $\frac{dh}{dt} = Q_i - Q_o$
- **Height evolution**, $c_2$: $h(t) = h(t - 1) + \int_{t-1}^{t} \dot{h}(\cdot) \cdot dt$
- **Output flow**, $c_3$: $Q_o = K_v u_c \sqrt{h}$

4 Dealing with temporal information in system description

Both DX and FDI communities have provided different approaches for dynamic systems modelling.

There is no general theory for CBD of dynamic systems [de Kleer, 2003]. Therefore, we build our discussion at the structural level, using previous works from the DX community [Dressler, 1996; Chantler et al., 1996].

Modelling of temporal information gives rise to two kind of constraints, assuming the model description is made up of a collection of first-order ordinary differential equations\footnote{This assumption does not constrain the presence of higher-order derivatives.} [Dressler, 1996; Chantler et al., 1996]:

- **Instantaneous constraints** – also known as intra-state –, for static relations; for instance \( (eq_1, v_1, v_2 \ldots) \);
- **Differential constraints** – also known as inter-state –, for dynamic behaviour; for instance \( (eq_2, x, \dot{x}) \), as in

\[
\dot{x} = \frac{dx}{dt}
\]

The FDI approach, using mainly numerical analytical models, has more standard specifications of dynamic aspects [Blanke et al., 2003], both in continuous and discrete time. 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. We focus our discussion on residual generation via ARRs obtained through structural analysis [Staroswiecki, 2002; Blanke et al., 2003].

Inclusion of temporal information in ARRs can be done in three different ways [Dustegör et al., 2006]. Difference between them comes from the way the relationship between an state variable, \( x \), and its derivative, \( \dot{x} \), is stated\footnote{Also called differential constraint in FDI terminology.}. These three methods have been compared in different works [Dustegör et al., 2006; Svärd and Wassén, 2006]. As can be seen, the approach which considers \( x \) and \( \dot{x} \) as different variables, and linked by the constraint \( (eq_1, x, \dot{x}) \), is equivalent to the DX approach. We shall focus our discussion in this kind of models, and ARR calculation algorithms based on Staroswiecki’s works (known as Lille method [Staroswiecki, 2002; Blanke et al., 2003]).

Differential constraints can be used for behaviour estimation in two ways, depending on the causal interpretation of the constraint: the derivative approach \( (\ddot{x}(t) = \frac{dx}{dt}) \) assumes the derivative can be computed based on present and past samples of \( x \), and the integral approach \( (x(t) = x(t-1) + \int_{t-1}^{t} \dot{x} \cdot dt) \) assumes the initial state \( x(t-1) \) is known. It has been demonstrated that both approaches have equivalent behaviour estimation capabilities for numerical models [Chantler et al., 1996], assuming adequate sampling rates and precise approximations were available, or assuming initial conditions are known.

Algorithms computing PCs or computing ARRs as in the Lille method can use differential constraints as expressed above. And both methods can include both types of causal interpretations [Pulido and Alonso, 2000; Staroswiecki, 2002] – looking for integral or derivative causality in the matching performed in the Lille method –. Using integral or derivative matchings for each differential constraint provides different computable models for each MEM and ARRs with different causal matchings, but impose no restriction in the way MECs and ARRs are computed, because both analysis are performed at the structural level (see [Pulido et al., 2007] for more details on this comparison).

One final issue must be addressed. The presence of cycles can halt local propagation [Katsillis and Chantler, 1997] while using GDE. Then, it will need an inference engine able to solve algebraic loops. For off-line dependency-recording this step can be done off-line [Pulido and Alonso, 2000; Staroswiecki, 2002].

**Cyclical structures containing differential constraints** To solve cyclical structures both algorithms, PCs calculation and ARR computation in the Lille method, follow identical approaches, coming respectively from DX and FDI points of view:

Cycles containing both instantaneous and differential constraints must be studied:

- using integral causality, loops containing differential constraints are not loops [Dressler, 1996], but spirals, because \( x \) and \( \dot{x} \) have different temporal indices;
- using derivative causality, no loop including \( x \) and \( \dot{x} \) can be solved. Hence, these loops in ARRs or MEMs must be rejected [Blanke et al., 2003].

Summarizing, both FDI by means of ARRs and Consistency-based diagnosis allow derivative and integral approaches. While DX approaches have opted by simulation
techniques –relying mainly in qualitative models–, traditionally the FDI community has opted for numerical models, and has rejected simulation; most FDI methods rely upon derivative estimation [Blanke et al., 2003], which has instead problems related with noise, disturbances, and parameter uncertainty.

4.1 PCs and ARRs for dynamic systems

Proposition 1 Given equivalent system descriptions including differential constraints for PCs and ARR calculation, the set of PCs and the complete set of minimal ARRs have same isolation capabilities.

Proof: Algorithms finding possible conflicts perform an exhaustive search for the complete set of minimally overdetemined subsystems, i.e. MECs. Pulido et al. have demonstrated [Pulido and Alonso-González, 2004] that both approaches were equivalent for static systems, if algorithms used for ARR computation provided a complete set of minimal conflicts. Moreover, only differential constraints have been added, and they are used for propagation the same way instantaneous constraints were used. Finally, both approaches remove illegal cyclical structures. Therefore, both will provide same results in terms of fault isolability.

Main difference between both approaches is the use of integral or derivative approaches: simulation of MEMs in the PC approach, and using ARR for estimation.

Previous results can be extended to minimal conflicts, ARRs, and the set of MEMs provided by the set of PCs, if the GDE inference engine is able to handle loops.

This proposition can not be easily extended for fault detection purposes. In the case of non-linear models, there is no guarantee for equivalence among results coming from every MEM associated to a MEC [Pulido and Alonso-González, 2004]. The same can be said about different ARRs associated to the same support-set [Cordier et al., 2004]. In that case, evaluations of different MEMs or ARRs can provide different results for fault detection purposes.

We will illustrate these concepts on our running example.

Based on the model description we have found the set of Possible Conflicts shown in table 1, which are minimal w.r.t. the set of constraints in their models. Moreover, in the plant we have considered the set of fault modes shown in table 2. PCs related to those fault modes can be seen in the Theoretically possible conflicts.

Table 1: PCs found for the plant; constraints, components, and the estimated variable for each possible conflict.

PV 1
c1, c2, c3, c4, c5, c7
T, V, C, Qobs, hobs
h
PV 2
c1, c2, c3, c4, c6, c7
T, V, C, Qobs, hobs
Qobs
PV 3
C1, c2, c5, c6, c7
T, Qobs, Qobs, hobs
h
PV 4
C1, c4, c5, c6
V, C, Qobs, hobs
Qobs

Table 3: PCs and their related fault modes.

f1 T Leakage in tank
f2 V Valve constant failure
f3 hobs Level sensor failure
f4 Qobs Output flow sensor failure

Table 2: Fault modes considered.

Figure 2 shows the MEM for PC1. In the and-or graph it can be seen that the height of the tank and its derivative, h(t + 1) and h′(t), are related through a differential constraint, and there is no loop because we are using integral approach.

Our proposal is to use only the structure of MEMs to find a way for estimating initial conditions for simulation by means of state observers without adding extra knowledge.

5 Using possible conflicts to design and integrate state observers

As it was mentioned in section 4, it is known that integral and derivative approaches can be equivalent for behaviour estimation with numerical models under given assumptions [Chantler et al., 1996]. Moreover, in the FDI community simulation, estimation, and state observers are also equivalent from a structural point of view, and for linear models [Gertler, 1998]. In fact, parity- and observer-based approaches provide residuals with similar structures: both of them use same measurable input and output signals, assume their structure and parameters are known, and they do not change. They differ in the way the input and output measurements are filtered. Additionally, comparing the structure of state-observers and parity equations, some authors have already proved that they can be equivalent [Magni and Mayon, 1994; Isermann, 2006].

The state-space form system description can be defined as:

\[ \dot{x}(t) = A \cdot x(t) + B \cdot u(t) + K_x \cdot (y(t) - C \cdot x(t)) \quad (1) \]
\[ y(t) = C \cdot x(t) \quad (2) \]

Additionally, \( \lim_{t \to \infty} e_y(t) = 0 \) and \( e_y(t) = y(t) - \hat{y}(t) \)

Depending on the selected value for the gain \( K \), we obtain: from simulation for \( K = 0 \), to prediction for \( A = K \cdot C \) [Puig et al., 2006].

Table 4: PCs found for the plant using derivative causality.

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), September 22–24, 2008, Blue Mountains, NSW, Australia.
Table 5: PCs and their related fault modes using derivative causality.

<table>
<thead>
<tr>
<th>PC</th>
<th>f₁</th>
<th>f₂</th>
<th>f₃</th>
<th>f₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC₁</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PC₂</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PC₃</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: System Description for the polybox example.

When differential constraints are included, algorithms used to find PCs provide interesting results if integral causality is used: the structure—a set of constraints—of a Minimal Evaluable Model can be implemented as a simulator or as a state-observer.

**Proposition 2** Those MEMs containing a state variable can provide the minimal structural description for a state-observer, if there exists an instantaneous constraint between the estimated state variable and its observed value.

This result comes from the way error \( e_y(t) \) is introduced in the generalized state-observer scheme in equations 1 and 2:

\[
e_y(t) = y(t) - \hat{y}(t) = y(t) - C \cdot \bar{x}(t)
\]

Error \( e_y(t) \) can produce a discrepancy between an observed variable, \( y(t) \), and \( C \cdot \bar{x}(t) \), which is a straight computation for that observed variable given a state-variable. Moreover, if the structure of the state-observer is minimal w.r.t. the set of constraints in the model, there could be no other discrepancy source within the observer.

By definition, a MEM is the minimal set of constraints needed to compute a discrepancy. No other discrepancy node is allowed (otherwise, it will not be minimal).

Algorithms used to find PCs are complete. They found every MEM for every ME in System Description. As a consequence, they are capable to find every causal path to estimate a state variable (using derivative, integration or algebraic equations). Hence, it can be traced backward if there is an instantaneous and-or arc—valid causal assignment—linking the observed variable with the state variable. If that arc exists, it will be the discrepancy node in the MEM, because it is minimal, and it can be used to compute \( e_y(t) \) if the MEM is implemented as an state-observer.

Let’s explain these ideas with the MEM found for \( PC₁ \) in figure 2. In that MEM, \( h(t) \) is the state variable, and \( h_{obs}(t) \) is its corresponding observed value. If \( h(t) - h_{obs}(t) \) is the discrepancy node in the MEM. Moreover, their difference is \( e_y(t) \), which can be used if the MEM is implemented as a state-observer, as can be seen in figure 3. Both subsystems represent a MEM for \( PC₁ \).

In figure 3 there is only one integration step for the sake of readability. If no discrepancy is detected, the corrected value of \( h(t+1) \) can be used to estimate \( Q_c(t+1) \), and \( U_c(t+1) \).

No additional information is required in the model description to provide these results. This search step must be included at the end of the algorithms used for analyzing cyclical structures in the Minimal Evaluable Models (and-or graphs) related to a Possible Conflict. This is an advantage against other approaches, which need an explicit equation to derive the structure for an state-observer [Blanke et al., 2003].
It is clear that there are two different stages in state-observers design:

1. Finding the structure (set of equations involved) of the observer.

This work is focused on providing the collection of minimal expressions for state-observers according to equation 1. No restriction is imposed in how the state-observer is implemented. A non-linear state-observer can be devised, for instance, using a linearization model + a Luenberger observer, or an Extended Luenberger observer, or an Extended Kalman Filter. Afterwards, it is necessary to analyze these equations for convergence and robustness issues.

The reader can find additional details on structural design of state-observers using PCs in [Pulido et al., 2007].

### 5.2 Integration proposal: increasing robustness with state observers

Using that ability to implement some MEMs as simulators or state-observers, we can improve the semi-closed loop simulation approach used by CBD using PCs, which was the main goal for this work.

State-observers has been used thoroughly in the FDI community [Gertler, 1998; Patton et al., 2000; Blanke et al., 2003]. They are able to generate a state-variable estimation, without fault, with noise in sensors and small parameter disturbances. If the state variable can be observed, it can be used to perform fault detection. Their main drawbacks are:

- small persistence for activated residuals, small activation time for faults, with noise in sensors and small parameter disturbances.

On the other hand, simulation in an interval, $\Delta t$, and using a dissimilarity comparison, DTW, in the interval, has different detection capabilities, being less sensible to noise in measurements. Semi-closed loop simulation iteratively introduces observations for initial conditions when the simulation interval elapsed. If there is no observations for some initial conditions, then it is necessary to use estimated values from simulation.

Our proposal is the integration of MEMs which can be implemented as state observers within the CBD framework with Possible Conflicts, because these observers will improve the estimations for the different states of the Possible Conflicts without fault, and they will not introduce additional fault isolation knowledge.

Running both MEMs in parallel, and assuming there is no fault detection, the state estimation given by the state-observer can be used as the initial state for the possible conflict simulation, as can be seen in figure 4. Every time the simulation interval $\Delta t$ elapses for the simulated MEM elapses, the state variable estimated by the state-observer, $EST_s$, is used as initial conditions for the next simulation period. This initialization process halts whenever a fault detection is done (due to a dissimilarity value between observed and simulated values, or due to a fault detection given by the state-observer).

This integration proposal gives flexibility to our consistency-based diagnosis system at the fault detection step, while fault isolation step is not altered (it is the same as described in [Pulido and Alonso, 2000] because no additional structural information outside the PC is introduced by the state-observer). Therefore, the decision logic in fault detection can be adjusted giving more weight to different factors:

- earlier fault detection time – fault detection by the state-observer is allowed–,
- low rate for false alarms – fault detection happens only when the simulation period has elapsed, and the information provided by the state-observer is used to confirm the detection and to provide more accurate detection time–,
- level on noise or parameter uncertainty, and so on.

### 6 Results on the case study

The study was made on a data-set, made up of examples obtained from several simulations for each fault mode
in the plant. Models and simulations were developed using Matlab©. In these simulations we introduced noise in the measurements (5%), and model uncertainties (5%). Each simulation lasted 1000 seconds, and contained several changes in the reference level of the tank, thus faults will occur most of the time in a transient state. We randomly generated fault magnitudes at different time instants within the interval [420, 470], being the simulation period \( \Delta = 60 \). The data sampling was 10 data per second.

We have reduced the mean and maximum values of errors integrating state observers within the Possible Conflicts computation in non-faulty situations, as can be seen in table 7, with 5% on sensor noise, and 5% on parameters disturbances. Therefore, we have increased fault detection sensibility.

Table 7: Mean and maximum error reduction combining PCs, \( PC_1 \), and state-observers, \( EST_1 \), for different simulation periods \( \Delta \). \( PC_4 \) does not admit state-observer implementation.

<table>
<thead>
<tr>
<th>( \Delta )</th>
<th>Mean</th>
<th>Max.</th>
<th>Mean</th>
<th>Max.</th>
<th>Mean</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>9.16</td>
<td>32.92</td>
<td>49.56</td>
<td>58.48</td>
<td>22.93</td>
<td>52.42</td>
</tr>
<tr>
<td>60</td>
<td>5.82</td>
<td>23.78</td>
<td>40.15</td>
<td>55.99</td>
<td>21.54</td>
<td>51.82</td>
</tr>
</tbody>
</table>

In table 8 and 9, detection time for different times fault occurrences are shown for PCs, State Observers, and the integration of both. Faults are pipe blockages of 10% and 30%, with 5% on sensor noise, and 5% on parameters disturbances. It can be seen the effect of the integration, improving detection rates and times.

Similar or better results have been obtained for the remaining set of faults, and specially for sensor faults. A complete description of the experimental study and the results can be found in [Bregón, 2007].

Table 8: Activation time for 10% pipe blockage. \( fp \) is a false positive: detection done before fault occurrence. \( no \) is a false negative: no detection. \( PC_3 \) is not affected by these faults. \( PC_1 \) represent the simulated \( MEM_1 \), \( EST_1 \) represent the state-observer for \( PC_1 \). \( INT_1 \) means that both the state-observer and simulated \( MEM_1 \) were used as proposed in section 5.2.

<table>
<thead>
<tr>
<th>Fault arises a ( t = )</th>
<th>( PC_1 )</th>
<th>( PC_2 )</th>
<th>( EST_1 )</th>
<th>( INT_1 )</th>
<th>( PC_2 )</th>
<th>( EST_2 )</th>
<th>( INT_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>420</td>
<td>430</td>
<td>440</td>
<td>450</td>
<td>460</td>
<td>470</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PC_1 )</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( EST_1 )</td>
<td>fp</td>
<td>646</td>
<td>fp</td>
<td>647</td>
<td>660</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( INT_1 )</td>
<td>540</td>
<td>540</td>
<td>490</td>
<td>540</td>
<td>540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( PC_2 )</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>fp</td>
<td>540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( EST_2 )</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( INT_2 )</td>
<td>480</td>
<td>540</td>
<td>540</td>
<td>480</td>
<td>480</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Activation time for 30% pipe blockage. \( fp \): false positive. \( no \): false negative. \( PC_3 \) is not affected by these faults.

7 Conclusions

PCs and ARRs exhibit identical fault isolation capabilities for dynamic systems described as a set of first order ODE equations, thus extending the BRIDGE framework for that kind of dynamic systems. The comparison has been made with the Lille method for ARR computation [Staroswiecki, 2002; Blanke et al., 2003]. Additional comparison is needed regarding different algorithms for ARR calculation allowing temporal information [Dustegör et al., 2006; Krysander et al., 2008].

Besides, parity-space and state-observers approaches in the FDI field [Gertler, 1998; Blanke et al., 2003; Isermann, 2006] are equivalent in terms of fault isolation capabilities, given that ARRs and state-observers can have identical structures [Magni and Mayon, 1994]. In this work, we proposed to use algorithms for computing PCs as a tool for state-observer design, while obtaining the structure of MEMs – i.e. the computational form for a Possible Conflict–, which can be implemented as an state-observer without including additional constraints in the model, which is an improvement w.r.t. [Staroswiecki, 2002]. The algorithms only provide the observer structure, but not the automatic synthesis based on such structure, as in [Christophe et al., 2004; Svärd and Wassén, 2006].

Moreover, we do not impose any restriction in the observer implementation (just its structure; in our test case, an EKF was used). Since, there are many results in the FDI community for automatic state-observer analysis and design, our approach can benefit from those results in the future.

Different MEMs for a conflict can provide different detection capabilities for non-linear systems. We have proposed a simple integration scheme for two implementations of a MEM in a PC, if possible: use an state-observer for initial state estimation, then use the estimation for a semi-closed loop simulation. Decision logic for fault detection can be tailored for each system, to get desired detection or false alarm rates.

Results on simulation are better with the integration of both methods. For small faults (10%), we can see that the integration proposal has better fault detection capabilities, against isolated simulation and state-observers. For medium to large faults, detection time is equal or smaller using the integration approach (as a result of a better estimation of initial values in each \( \Delta \) simulation period), and detection time provided by state-observers. We are now testing this approach in more complex real systems with unknown noise and uncertainties. In this case, the state-observer gain must be even more carefully designed.

Integration of state-observers for fault detection with consistency-based diagnosis was done in TRANSCEND [Mosterman and Biswas, 1999], where Bond-Graphs were used to derive both state-observers Temporal Causal Graphs used in consistency-based diagnosis. Then both TRANSCEND and PCs provide a systematic method to derive state-observers, and causal structures for fault isolation.
But there are several differences. In the PC approach, the causal structure for the state-observer and the simulated MEM is the same, but additionally, minimality is guaranteed. Another difference is that MEMs are obtained offline following ATMS+GDE approach (not ARR computation). There could be different causal structures [Cordier et al., 2004; Pulido and Alonso-González, 2004] because minimality is required, and conflicts can be found as one estimation against one measurement, or two estimations for same variable. Moreover, both approaches use state-observers and consistency-based diagnosis in slight different manners: using PCs, fault detection and isolation runs both the state-observer and simulation MEMs in parallel, and the state-observer gives an initial state for simulation in absence of faults; in TRANSCEND state-observers are only used for fault detection, which feeds the fault isolation step when a detection is done. Finally, the causal structure for the MEM is pre-compiled, with no propagation back and forward in the causal structure, as it is done in TRANSCEND.

Acknowledgments: This work was supported by the Spanish Ministry of Education and Culture (MEC 2005-08498).

References


Design requirements for the diagnosability of distributed discrete event systems

Pauline Ribot, Yannick Pencolé and Michel Combacau
LAAS-CNRS, Université de Toulouse
7 avenue du Colonel Roche, F-31077 Toulouse, France
{pribot,ypencole,combacau}@laas.fr

Abstract

We address the problem of fault diagnosability in distributed discrete-event systems. Previous works mainly propose different ways to check whether a fault is diagnosable or not. Nowadays, due to the complexity of the engineered systems, this check is not enough and a better feedback is required in order to redesign and guarantee the diagnosability of a fault. This paper defines the problem of the automatic computation of design requirements for the diagnosability of distributed discrete event system as a cost optimization problem.

1 Introduction

We address the problem of fault diagnosis in distributed discrete-event systems. It consists in determining the occurrences of fault events from observations and system knowledge. This problem has been studied for many years, [Sampath et al., 1995], [Debouk et al., 2002], [Console et al., 2002], [Lamperti and Zanella, 2003]. In these previous works, the objective is to model the system and to apply monitoring algorithms on it but they all share the same weakness: none of them takes into account any diagnosability issue about the system. If a diagnosability analysis is performed on the system, then the diagnosis algorithm is more efficient and less costly because new information is taken into account before implementing it.

In this paper, we adopt the following point of view. Due to the complexity of new large systems and the new requirements such as maintenance, reliability or security, the diagnosability study must be performed at the design stage to specify a system or to modify its specification. One difficulty in the design of a distributed system is that it is usually conceived by different designers which are in charge of only a part of the system. It follows that the integration of the different parts of the system is very complex and has a direct impact on the difficulty to guarantee the diagnosability objective of the whole system.

Our aim is to design a monitoring and diagnostic architecture for the maintenance of aeronautical systems. For this purpose, we try to determine design requirements of the distributed system as modifications which could be useful for the different designers to improve and to guarantee some diagnosability objectives of the whole system. This analysis relies on both the design of the distributed system itself (design and integration of components and their communication) and the design of the monitoring architecture in charge of diagnosing the system. We propose to formalize this problem as a cost optimization problem in a distributed framework. We then show the relationship between optimizing the cost of the design of the distributed system and optimizing the choice for a suitable monitoring architecture.

The paper is organized as follows. Section 2 recalls formal background on fault diagnosis and diagnosability. Section 3 defines the optimization cost problem for guaranteeing diagnosability in a distributed discrete-event system. Section 4 introduces a methodology that determines, given a pre-specified system and a set of observations, a set of design requirements for diagnosability that takes into account the inherent characteristics of a distributed system and minimizes the implementation cost.

2 Background

2.1 Formal definition of the DES formalism

Our study takes place in a discrete event system framework for model-based diagnosis as defined in [Sampath et al., 1995]. This framework has been developed for several years and is used for different application types (communication networks, business processes, middleware, ...).

Let us consider a distributed system $\Gamma$ composed of interacting components $\Gamma = \{ \Gamma_1, \Gamma_2, \ldots, \Gamma_n \}$ which evolve by the occurrence of events. We suppose that we already have a specification of this distributed system with an initial analysis of its behavior in the case where faults occur. This analysis is provided by designers, such as FMEA (Failure Mode Effect Analysis) data for example. The distributed system can then be modelled as a set of finite state machines (see Figure 1), each finite state machine (FSM) representing the model of a component (i.e. a local model). Depending on the context, $\Gamma$ shall denote the system or its model.

Definition 1 (Local model) A local model $\Gamma_i$ is a FSM $\Gamma_i = (Q_i, \Sigma_i, T_i, q_{0i})$ where:

- $Q_i$ is a finite set of states;
The language $L(\gamma)$ is generated by the FSM $\|\gamma\|$ obtained from the classical synchronization operation, denoted $\|$ and whose definition can be found in [Pencolé et al., 2006]. The FSM $\|\gamma\|$ results from the synchronized product of $m$ component models on the set of interactive events $\Sigma_{\gamma} = \bigcup_{i=1}^{m} \Sigma_{i}$. The behavior of the system is explicitly represented by $\|\Gamma\|$ (also called the global model in [Sampath et al., 1995]). In the following, we will suppose that the components of the system are live and the system is deadlock-free. The consideration of problems directly linked to blocked states is not in the classical theory. New definitions and adhoc resolutions are required but they are not the topic of this paper.

The second assumption is that any model $\Gamma_i$ is globally consistent. Before defining the global consistency of a local model, we need to introduce the projection operation. The projection operation can be recursively defined as follows. Let $\epsilon$ denote the empty sequence in $\Sigma^*$.  

**Definition 2 (Projection)** The projection operation $P_{\Sigma^*}$:  

$\Sigma^* \rightarrow \Sigma^*$ is such that $P_{\Sigma^*}(\epsilon) = \epsilon$ and for all $uv \in \Sigma^*$, $u \in \Sigma$,  

$$P_{\Sigma^*}(uv) = \begin{cases} uP_{\Sigma^*}(v) & \text{if } u \in \Sigma' \\ P_{\Sigma^*}(v) & \text{otherwise.} \end{cases}$$

**Definition 3 (Global consistency)** A model $\Gamma_i$ is globally consistent if the following condition holds:  

$$L(\Gamma_i) = P_{\Sigma_{\gamma_i}}(L(\Gamma_0) || \Gamma_1 || \ldots || \Gamma_n)).$$

A model $\Gamma_i$ is globally consistent if every transition defined from any state of the model is fired in a global behavior of the system. Given any model of $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$, it is always possible to obtain an equivalent set of models that are globally consistent using techniques like in [Su and Wonham, 2004] for instance.

### 2.2 Fault diagnosis on a subsystem

The diagnosis problem is similar to the one specified in [Pencolé et al., 2006]. The problem resolution requires the computation of a set of diagnosers. Let $F$ be a fault event occurring on a component $\Gamma_i$ which belongs to a subsystem $\gamma$, a diagnoser of the subsystem $\gamma$ is in charge of diagnosing $F$. In the following, $F \in w$ denotes the occurrence of $F$ in the event sequence $w$. After any sequence of observations $\sigma$ emitted by $\gamma$, the diagnoser is a function that provides a diagnosis information $\Delta_\gamma(F, \sigma)$ which is one of the following three types.

- $\Delta_\gamma(F, \sigma) = F$-sure if $\forall w \in L(\gamma)$ such that $P_{\Sigma_{\gamma_i}}(w) = \sigma, F \in w$.
- $\Delta_\gamma(F, \sigma) = F$-safe if $\forall w \in L(\gamma)$ such that $P_{\Sigma_{\gamma_i}}(w) = \sigma, F \notin w$.
- $\Delta_\gamma(F, \sigma) = F$-ambiguous if $\exists w, v \in L(\gamma)$ such that $P_{\Sigma_{\gamma_i}}(w) = \sigma$ and $P_{\Sigma_{\gamma_i}}(v) = \sigma, F \in w$ but $F \notin v$.

The global diagnosis is given by $\{\Delta_\Gamma(F, \sigma), F \in \Gamma_f\}$. We represent the diagnoser function as a deterministic FSM $\Delta_\gamma(F)$. This machine is built from the projection of the model $\|\gamma\|$ on the observable events in $\Sigma_{\gamma_i}$. Then the diagnoser describes the observable behavior of $\gamma$. The $F$-diagnoser $\Delta_\gamma(F)$ can be formally defined as follows.

**Definition 4 (F-Diagnoser of a subsystem $\gamma$)** Given an observation set $\Sigma_{\gamma_i}$, the $F$-diagnoser $\Delta_\gamma(F)$ of a subsystem $\gamma$ in charge of diagnosing a fault $F$ is a deterministic finite state machine $\Delta_\gamma(F) = (Q_{\gamma_i}, \Sigma_{\gamma_i}, T_{\gamma_i}, q_{\gamma_i}^0)$ where:

1. $Q_{\gamma_i} \subseteq 2^{|Q_i|} \times \{\sigma, \delta\}$ is the set of states which contain the diagnosis information;
2. $\Sigma_{\gamma_i} = \Sigma_i$ is the set of events;
3. $T_{\gamma_i} \subseteq Q_{\gamma_i} \times \Sigma_{\gamma_i} \times Q_{\gamma_i}$ is the set of transitions;
4. $q_{\gamma_i}^0 = (q_0, \emptyset)$ is the initial state.
The transitions of $T_{d\gamma}$ are the transitions $q_{d\gamma} \xrightarrow{\varepsilon} q'_{d\gamma}$ reachable from the initial state $q_{d\gamma0} = \{(q_0, \emptyset)\}$, with $q_{d\gamma} = \{(q_1, f_1), \ldots, (q_m, f_m)\}$ and $q'_{d\gamma}$ is the set \{(q'_1, f'_1), \ldots, (q'_m, f'_m)\}, such that for any $(q_j, f_j) \in q_{d\gamma}$, there exists a transition sequence $q_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} \ldots \xrightarrow{u_p} x_p \xrightarrow{\varepsilon} q'_j$ in the model $\|\gamma\|$ with $u_k \in \sum_{q_k} \cup \sum_{f_k}$, $\forall k \in \{1, \ldots, p\}$, $\forall j \in \{1, \ldots, m\}$ and $f' = f \cup \{(F) \cap \{u_1, \ldots, u_p\}\}$.

Two examples of diagnosers are illustrated in Figure 2.

We note $Diag(q_{d\gamma})$ the diagnosis function associated to the state $q_{d\gamma}$:

1. $Diag(q_{d\gamma}) = F$-safe if $\forall (q_i, f_i) \in q_{d\gamma}$, $f_i = \{F\}$;
2. $Diag(q_{d\gamma}) = F$-sure if $\forall (q_i, f_i) \in q_{d\gamma}$, $f_i = \emptyset$;
3. $Diag(q_{d\gamma}) = F$-ambiguous otherwise.

2.3 Diagnosability

Diagnosability defines a property that measures the ability of the monitoring system to diagnose faults occurring in the supervised system. We rephrase the diagnosability definition from [Sampath et al., 1995] and [Pencolé, 2005].

**Definition 5 (Global diagnosability)** The fault event $F$ is globally diagnosable in a system $\Gamma$ iff

$$\exists l \in \mathbb{N}, \forall u, v \text{ such that } |P_{\Sigma_u}(v)| \geq l \Rightarrow \Delta_{\Gamma}(F, \sigma) = F\text{-sure},$$

(1)

where $u$ is an event sequence in $\Gamma$ from the initial state $\emptyset$, ending with the occurrence of the fault event $F$ to a state $q_1$, $v$ is an event sequence in $\Gamma$ from $q_1$ and $\sigma$ is the observable sequence produced by the event sequence $uv$ (i.e. $\sigma = P_{\Sigma_u}(uv)$).

Several tools allow the checking of diagnosability [Sampath et al., 1995], [Jiang et al., 2001], [Yoo and Lafortune, 2002].

3 Design for Diagnosability

The objective is to establish requirements to designers to ensure the diagnosability of the designed parts of the distributed system. Two different ways can be considered to reach the objective. The first one consists in providing requirements for the design of a diagnosable system (diagnosability analysis is performed at the design stage of the system) and the second one relies on a feedback for redesigning the system and increasing the system diagnosability. Both issues are related to the problem of assistance to the design like in [Pencolé, 2005]. In this paper, we focus on the first approach.

3.1 General requirements for diagnosability

Global diagnosability implies the use of a global centralized diagnostic architecture on the whole system which may not be always feasible in practice. Moreover the global diagnosability checking requires the computation of the global model $\|\Gamma\|$ which is very complex and also not always feasible. We therefore prefer to reason locally to subsystems in order to remove this limitation for large distributed systems. We want to determine subsystems that are sufficient to monitor in order to diagnose an anticipated fault occurring on a component. We introduce the definition of local diagnosability based on the fault diagnosis on a subsystem $\gamma$ (see Section 2.2). This definition can be applied to any subsystem $\gamma$.

**Definition 6 (Local diagnosability)** The fault event $F$ is locally diagnosable in a subsystem $\gamma$ iff

$$\exists l \in \mathbb{N}, \forall u, v \text{ such that } |P_{\Sigma_u}(v)| \geq l \Rightarrow \Delta_{\gamma}(F, \sigma) = F\text{-sure},$$

(2)

where $u$ is an event sequence in $\gamma$ from the initial state $q_0$, ending with the occurrence of the fault event $F$ to a state $q_f$, $v$ is an event sequence in $\gamma$ from $q_f$ and $\sigma$ is the observable sequence produced by the event sequence $uv$ (i.e. $\sigma = P_{\Sigma_u}(uv)$).

Global diagnosability corresponds to local diagnosability on the whole system $\Gamma$ [Pencolé, 2004].

The second requirement is that the observability of the whole system has to be live (i.e. $P_{\Sigma_u}(L(\Gamma))$ is live) as explained in [Debouk et al., 2002]. In practice, if we want to diagnose a fault $F$ on a subsystem $\gamma$, we have to specify this condition locally by the observation fairness of the system. The observability of a system is globally fair when the observability of each component is globally fair.

**Definition 7 (Observation fairness)** Observation fairness means that any observable component of the system will always emit observations after a finite number of events.

As each observable component is live, observation fairness holds if there is no cycle of unobservable events in the component. It follows that for a subsystem $\gamma$, each event sequence in $L(\gamma)$ must always be continued by a finite event sequence which ends with an observable event of $\gamma$: $\forall w \in L(\gamma)$, $\exists p \in \mathbb{N}$ such that $\forall uw' \in L(\gamma)$, $|w'| = p, \exists \sigma \in \sum^*_{\gamma} \setminus \{\epsilon\}, P_{\Sigma_{\sigma}}(uw') = P_{\Sigma_{\sigma}}(w).\sigma$.

The following property states the relationship between local and global diagnosability [Pencolé, 2004].

**Property 1** Under the assumption of observation fairness, if a fault $F$ is locally diagnosable on a subsystem then $F$ is globally diagnosable.

By this property, it may be unnecessary to observe the whole system to diagnose a fault $F$ occurring on a component of $\Gamma$. Observing only a subsystem $\gamma$ can thus be sufficient.

To ensure diagnosability of the system, some modification operations have to be considered depending on the monitoring architecture. These operations consist in increasing the subsystem observability.

3.2 Types of modifications for the system design

The first operation consists in increasing the observability by selecting types, location and number of sensors. The added sensors are supposed to be reliable and without noise. This operation type tends to be similar to a sensor selection problem like in [Debouk et al., 1999] or [Jiang et al., 2003]. The second operation acts directly on the behavior of the subsystem. Based on the specification of a system (preexistent models), some possible operations are enumerated hereafter. All these modifications are followed by their physical significance.
1. Observing an event \( e \) (that may be a fault) which occurs in a component of the subsystem \( \gamma \): \( e \in \Sigma_{\ell \gamma} \cap \Sigma_{o \gamma} \). Making an event observable means to add a sensor that is able to detect the occurrence of such an event on a component in the studied subsystem.

2. Observing an event \( e \) which occurs on an other component \( \alpha \) which interacts with \( \gamma \): \( e \in \Sigma_{\ell \gamma} \cap \Sigma_{o \alpha} \), where \( \alpha \notin \gamma \). To observe an event on a component which interacts with the subsystem, we need to place a sensor on this component and to consider a communication protocol in the diagnostic architecture.

3. Observing an interactive event \( i \) between the subsystem \( \gamma \) and an other component \( \alpha \): \( i \in \Sigma_{\ell \gamma} \cap \Sigma_{o \alpha} \cap \Sigma_{o \alpha} \cap \Sigma_{o \alpha} \), where \( \alpha \notin \gamma \). To make an interactive event observable consists, for example, in putting a sensor on the communication bus between two components or on a middleware.

4. Observing an event \( e \in \Sigma_{\ell \gamma} \) only if given conditions \( cond \) are verified by introducing two new events \( cond e \in \Sigma_{\gamma} \) and \( \neg cond \neg e \notin \Sigma_{\gamma} \). To observe an event in a given condition we need to have a specific sensor which can be controlled on-line depending on other information. This sensor allows an active acquisition of information like in [Thorsley and Teneketzis, 2007].

5. Adding/Deleting a transition \( t \) in the model of \( \gamma \). New sensors can be generated like an alarm sensor after a given sequence of observations for example or new protocols can be implemented (communication protocol for instance). Transitions are reorganized by this new implementation and some of them may be deleted.

To each modification on the system is associated a cost which is known and listed in a modification dictionary. For example, a sensor cost depends on the type of the sensor, a pressure sensor would be more expensive than a temperature sensor. The cost of all modifications performed on the system is denoted \( C_D \). Some modifications are infeasible. An infinite cost is associated to such modifications like for example the observation of some fault events.

### 3.3 Specification of the diagnostic architecture

Having sensors is not enough to ensure diagnosability. A diagnostic architecture using these sensors must be deployed upon them and have access to them. Accessing to these information resources induces a cost \( C_M \) for the monitoring system that depends on the diagnostic architecture type. The goal is to make the system diagnosable by optimizing the cost of modifications listed above (for example the cost associated to the sensor placement) but also the cost of the monitoring system (algorithm, computational resources, sensors utilization).

There mainly exist three different types of diagnostic architecture. In a centralized diagnostic architecture like in [Sam-path et al., 1995], all information from the components are centralized in the same place. In this architecture type, the data analysis is complex because all data are collected by the monitoring system without any processing. A lot of memory resources and communications are necessary for the monitoring system and induce the cost \( C_M \). A centralized architecture may be not well suited for distributed systems. For large distributed systems it is more natural to adopt a decentralized or a distributed diagnostic architecture. In a decentralized diagnostic architecture, diagnosis decisions are sent from local diagnosers, which do not necessary communicate with each other. These decisions are then merged by a coordinator in order to establish a global diagnosis and to allow to make global decisions about the whole system. In a distributed architecture, there is a diagnoser per component or per subsystem that performs its own diagnosis. In order to get globally consistent diagnoses, the diagnosers need to exchange a lot of information. This communication between the diagnosers induces a bandwidth cost which is included in the cost of the monitoring algorithm \( C_M \).

### 3.4 Problem formalization like a cost optimization problem

The goal is to give some requirements to the designers about how to make a system diagnosable by minimizing the total cost of operations on the system, denoted \( C_D \), and minimizing the cost of the monitoring system algorithm \( C_M \):

\[
C_G = \min \sum_{i=1}^{p} (C_{D_i} + C_{M_i}),
\]

where \( p \) is the number of faults that may occur in the system and for which diagnosability must be guaranteed.

The challenge is to determine a trade-off between both costs (sensor placement and the cost of the associated monitoring architecture).

For example, one possible modification is to observe an event on another component in the neighbourhood. For this purpose, the monitoring system needs to use a communication protocol to get back the observation from the component. So we have to consider the cost \( C_M \) of the algorithmic procedure to communicate information to the monitoring system in addition to the cost directly associated to the sensor.

The second difficulty in the cost optimization problem is that all fault events \( F \in \Sigma_f \) must be considered. Since the components of the system communicate by interactive events from \( \Sigma \), it could be interesting to observe interactions even if this event type has a high cost because its observation could bring useful information to diagnose more than one fault event and thus reduces the monitoring cost.

### 4 Methodology

This section introduces a methodology that provides combining requirements and their costs for the designers. It relies on two properties: accuracy and monotony. We show how these properties can help to establish the methodology. In this section, the problem is restricted to operation of the non-active sensor placement. The only possible modification is the addition of observable events (operations from 1. to 3. in Section 3.2).
4.1 Accuracy

Independently from the diagnosability of a fault $F$, finding a subsystem whose diagnosis is accurate is interesting because it is a way to bound the size of the subsystem to monitor and thus the cost of the monitoring algorithm $C_M$. The diagnosis of a subsystem $\gamma$ is said to be accurate if it is sufficient to observe it in order to provide a diagnosis which is consistent with the whole set of observations. Then the monitoring system does not require information from any other component. We are sure to have a diagnosis equivalent to the global diagnosis at any time.

Accuracy is formally defined as follows. Let $\Delta_\gamma(F)$ be the diagnoser of the subsystem $\gamma$ and $\sigma_\gamma = P_{\Sigma_{\gamma}}(\sigma)$ be the projection of the sequence $\sigma \in \Sigma_\gamma$ on the observable events of $\gamma$.

Definition 8. The diagnosis of a subsystem $\gamma$ is accurate for a fault event $F \in \Sigma_f$, iff the diagnoser $\Delta_\gamma(F)$ is such that

$$\forall \sigma \in \Sigma_{\sigma,\gamma}, \Delta_\gamma(F, \sigma) = \Delta_\gamma(F, \sigma_{\gamma}).$$

We can always find a subsystem whose diagnosis is accurate (diagnosis of the global system $\Gamma$ always accurate). The accuracy property is defined from the global diagnosis but there exists a sufficient local criterion to check it.

How to check accuracy

Diagnosis accuracy for a subsystem $\gamma$ can be checked from a finite state machine called an interactive diagnoser. An interactive diagnoser $\Delta^{int}_\gamma(F)$ is an extension of the diagnoser $\Delta_\gamma(F)$ defined in Section 2.2. Let us suppose an observable configuration $\Sigma_{\sigma,\gamma}$ for the subsystem $\gamma$, the interactive diagnoser $\Delta^{int}_\gamma(F)$ results from Definition 4 (see Figure 2).

The interactive diagnoser is represented by the quadruple $\Delta^{int}_\gamma(F) = (Q_{\delta_{\gamma}}, \Sigma_{\delta_{\gamma}}, T_{\delta_{\gamma}}, q_{\delta_{\gamma}0})$, where $\Sigma_{\delta_{\gamma}} = \Sigma_{\sigma,\gamma} \cup \Sigma_{\gamma}$. The set $\Sigma^{int}_{\gamma}$ represents the events of $\gamma$ which interact with the other components: $\Sigma^{int}_{\gamma} = \Sigma_{\sigma,\gamma} \cap \Sigma_{\sigma,\gamma'}$.

Like in the classical diagnoser, the states of $\Delta^{int}_\gamma(F)$ contain the diagnosis information. We recall that $Diag(q_{\delta_{\gamma}})$ denotes the diagnosis information contained in the state $q_{\delta_{\gamma}}$ of $\Delta^{int}_\gamma(F)$ which is one of the following three types: $F$-sure, $F$-safe, $F$-ambiguous. The determination of a subsystem with an accurate diagnosis relies on a criterion introduced in [Pencolé, 2005] that we recall in Property 2. The notation $q_i \xrightarrow{w} q_j$ means that there exists an event sequence $w$ from the state $q_i$ which leads to the state $q_j$ in $\Delta^{int}_\gamma(F)$. Let $S(\sigma_{\gamma})$ be the set of event sequences in $\Delta^{int}_\gamma(F)$ from $q_{\delta_{\gamma}0}$ whose observable part is exactly $\sigma_{\gamma} \in \Sigma_{\sigma,\gamma}$.

Property 2. The diagnosis of a subsystem $\gamma$ is accurate if the following criterion holds: $\forall \sigma_{\gamma}, q_{\delta_{\gamma}}, q'_{\delta_{\gamma}} \in Q_{\delta_{\gamma}}$, such as $q_{\delta_{\gamma}0} \xrightarrow{w} q_{\delta_{\gamma}}$ and $q_{\delta_{\gamma}0} \xrightarrow{w'} q'_{\delta_{\gamma}}$ with $w, w' \in S(\sigma_{\gamma})$, $Diag(q_{\delta_{\gamma}}) = Diag(q'_{\delta_{\gamma}})$.

Figure 2 presents the interactive diagnoser $\Delta^{int}_\gamma(F3)$ and an accurate $F3$-diagnoser defined on the subsystem $\gamma = \{\Gamma_3\}$ for $\Sigma_{\sigma,\gamma} = \{e5, s3\}$.

Any event sequence in $\Delta^{int}_\gamma(F3)$ projected on the observable events of $\Sigma_{\sigma,\gamma}$ leads to states which contain the same diagnosis information, which have the same label. For example, every path from the initial state emitting the observable sequence $s3e5s3e$ is $F3$-sure and every path emitting the sequence $e5s3e5s5$ is $F3$-ambiguous. Observations from other components could give information about the occurrence of the event $s4$ which is not observable but the diagnosis result provided by $\Delta^{int}_\gamma(F3)$ after the occurrence of $e5$ does not depend on the occurrence of $s4$. So in this case observations from other components cannot disambiguate the diagnosis.

How to make a subsystem accurate

By property 2, we notice that if all interactions of $\gamma$ with other components are observable the previous criterion holds. Then a simple way to make a system accurate is to observe the interactive events [Ribot et al., 2007]. Let $\Sigma^{int}_{\gamma}$ be the set of events of $\gamma$ that interact with the other components: $\Sigma^{int}_{\gamma} = \Sigma_{\sigma,\gamma} \cap \Sigma_{\sigma,\gamma'}$.

Property 3. Let $\gamma$ be a subsystem, the diagnoser $\Delta_\gamma(F)$ is $F$-accurate for all $F \in \Sigma_f$, if $\Sigma^{int}_{\gamma} \subseteq \Sigma_{\sigma,\gamma}$.

4.2 Property monotony

The methodology consists in determining a set of modifications that guarantee observability fairness, diagnosability and accuracy. It is interesting to know if one of these properties can be preserved after the modifications realized to guarantee the other ones.

Definition 9. (Monotony) Let $Prop$ be a boolean application ($\forall \%$, $Prop : P(\%)$ $\mapsto \{0, 1\}$, where $P(\%)$ represents...
the power set of \(\mathcal{X}\), \(Prop\) is monotonic iff

\[
\forall X, Y, X \subseteq Y \subseteq X, \ Prop(X) \Rightarrow Prop(Y).
\]

This definition shows that for two sets \(X, Y\), if \(X\) is included in \(Y\), any property which is verified by \(X\) is also verified by \(Y\). In our study case \(X, Y\) are two sets of observ-

able events \((X \subseteq \Sigma, Y \subseteq \Sigma)\). The monotonity of some prop-

erties like observability, normality or diagnosability has been studied in [Jiang et al., 2003]. The observation fairness (De-

finition 7) is obviously a monotonic property. If any observ-

able component of the system always emits an observation in a

finite delay, the addition of new observations preserves this

property.

Property 4 If \(F\) is diagnosable with an observation set \(\Sigma_o\) satisfying the observation fairness, then \(F\) is diagnosable with an observation set \(\Sigma_o' = \Sigma_o \cup \{o\}\).

Proof: By the negation of Definition 6 (Local diagnosa-

bility), a fault \(F\) is not diagnosable for an observation set \(\Sigma_o'\) if there exists an infinite sequence of observations \(\sigma\) such that the diagnosis \(\Delta_e(F, \sigma)\) is \(F\)-ambiguous: there exist at least two infinite event sequences \(p_1, p_2\) such that \(F\) is in \(\pi_1\), \(p_1\) and \(\pi_2\), \(p_2\) such that \(F\) is in \(\pi_1\) and \(\pi_2\) is not in \(\pi_2\), whose projection on the observations of \(\Sigma_o'\) is \(\sigma\). Let \(\Sigma_o\) be a new set of observations deprived of the event \(o\): \(\Sigma_o' = \Sigma_o \setminus \{o\}\). We prove that if \(F\) is not diagnosable with \(\Sigma_o'\), then \(F\) is not diagnosable with \(\Sigma_o\). If \(o \notin \Sigma\), we still have the infinite sequence \(\sigma\) such that the diagnosis is \(F\)-ambiguous. If \(o \in \Sigma\), as the observa-

bility configuration \(\Sigma\) satisfies the observability fairness (Definition 7), then we obtain a new infinite sequence \(\sigma_o\) of observations deprived of \(o\), such that the diagnosis is \(F\)-ambiguous as it is the projection of \(p_1\) and \(p_2\) on the observations of \(\Sigma_o\). \(\square\)

Property 5 Accuracy is not a monotonic property.

Proof: We have shown in Figure 2 that we have an accurate \(F3\)-diagnoser by observing \(\{e5, s3\}\) on \(\Gamma_3\). We consider a new observation \(e6\) on the component \(\Gamma_3\). Let \(\sigma_1\) and \(\sigma_2\) denote two global observable sequences in \(\Sigma_4\) such that \(\sigma_1 = e6e7e3e5\) and \(\sigma_2 = e6e4e5\). After observing \(\sigma_1\), the global diagnosis is \(F\)-safe and after \(\sigma_2\), the global diagnosis is \(F\)-sure. The projection of both event sequences \(\sigma_1\) and \(\sigma_2\) on events of \(\Gamma_3\) is \(e6e5\). Locally, \(\sigma_1\) and \(\sigma_2\) are not distinguishable and the local diagnosis after observing \(e6e5\) is \(F\)-ambiguous. The local diagnosis is not equiv-

alent to the global diagnosis, then the diagnosis of \(F3\) is not accur-

ate anymore by considering the observation \(e6\) on \(\Gamma_3\). \(\square\)

Accuracy is a non monotonic property but it can be pre-

erved in the case where all interactions of the subsystem \(\gamma\) with the other components are observable. In this specific case described by Property 2, accuracy still holds after the addition of a new observation.

Diagnosability property is always preserved by consider-

ing new observations whereas accuracy property is preserved only in some specific cases. The monotonity can help with the sensor choice by determining some selection criteria in order to preserve diagnosability and accuracy properties.

4.3 Algorithm

This section presents an algorithm that determines a sub-

system and proposes some modifications on it in order to make a fault occurring on one of its component with an accurate diagnosis and minimal costs. We consider a system of \(n\) components with an initial minimal observable configuration \(\Sigma_o\). The system may be specified with an initial set of sensors which are already available. The initial observable configuration may be empty (no sensors already placed on the system), this case is illustrated by an example in 4.4.

Algorithm 1 selects a subsystem for which the total cost \(C_G\) is minimal (between the supervision cost \(C_M\), the cost \(C_A\) to make it accurate and the cost \(C_D\) to make it diagnosable for the fault \(F\) in component \(\Gamma_i\)). It also returns the requirements \(Req\) as a set of modifications to perform on the subsystem. In this study, we consider modifications from the sensor placement (i.e. operations from types 1., 2. or 3. of Section 3.2). Since diagnosability is a monotonic property, we look for an observable configuration that makes the system diagnosable before considering accuracy. If the subsystem is diagnosable, the diagnosability property will be preserved by adding observable events to get accuracy. If the subsystem is not diagnosable, the optimization is performed simultaneously on both costs: \(C_A\) and \(C_D\).

In the proposed algorithm, the expression sub-

system\((\Gamma_i, k)\) denotes the set of subsystems com-

posed of \(k\) components and containing \(\Gamma_i\). We first need to determine if there is a solution to the sensor placement problem. The function \(SolutionExistence(\gamma, F)\) relies on the Property 6.

Property 6 If the fault event \(F\) occurring on a component of the subsystem \(\gamma\) is not diagnosable in \(\gamma\) by considering all events (except the fault events) as observable, then there is no solution for the sensor placement problem.

If all events are observable and \(F\) is not diagnosable, no event from other components which interacts with \(\gamma\) will provide more information to make \(F\) diagnosable. The only way to make \(F\) diagnosable in the component is then to redesign it, by considering operations of type 4. or 5. (see Section 3.2) which is out of the topic of this methodology, in order to have \(F\) diagnosable by considering the new set of all observations.

The function \(Monitoring\) induces a cost \(C_M\) for the mon-

itoring of a subsystem depending on the implementation of the diagnosis architecture as explained in 3.3. The functions \(CheckingAccuracy\) and \(CheckingDiagnosability\) are boolean. The first function applies the criterion of Property 2 to determine if the considered subsystem is accurate or not and the second one uses an algorithm to check the sub-

system diagnosability for a fault \(F\) as in [Jiang et al., 2001], [Pencolé, 2004]. These algorithms are polynomial in the number of states \(|\gamma|\). The functions \(MakeDiagnosable\) and \(MakeAccurate\) use modifications from types 1., 2. or 3. of Section 3.2 and techniques from sensor placement as in [Jiang et al., 2003] to obtain a diagnosable subsystem \(\gamma\). The
Algorithm 1 Requirements and costs for diagnosability

1: **Input:** $F \in \Sigma_{f_i}, \Gamma = \{\Gamma_1, \ldots, \Gamma_n\}, \Sigma_o$
2:  
3: \( \text{repeat} \)
4: \( C_G^0 = \infty \)
5: \( k \leftarrow 1; \text{Req} = \emptyset \)
6: \( \text{for all } y \in \text{subsystem}(\Gamma_i, k) \) do
7: \( \text{Go step 24} \)
8: \( \text{end if} \)
9: \( C_M = \text{MonitoringExistence}(\gamma, F) \)
10: \( C_D = 0; C_A = 0; \)
11: \( \text{if } \text{CheckDiagnosability}(\gamma, F) \) then
12: \( (\text{Req}, C_A) \leftarrow \text{MakeAccurate}(\gamma) \)
13: \( \text{end if} \)
14: else
15: \( (\text{Req}, C_A, C_D) \leftarrow \text{MakeDiagnosable}(\gamma, F) \) \( \wedge \) \text{MakeAccurate}(\gamma) \)
16: \text{end if} \)
17: \( C_G^k = C_M + C_A + C_D \)
18: \( C_G^k \leftarrow \min(C_G^k, C_G^{k-1}) \)
19: \text{end for} \)
20: \( k \leftarrow k + 1 \)
21: \text{until } (C_G^{k-1} \geq C_G^{k-2}) \wedge (k \geq 2) \wedge (k \leq n) \)
22: \( C_G \leftarrow C_G^{k-1} \)
23: \( \text{Go to step 25} \)
24: \text{Output: } \text{No solution} \)
25: \text{Output: } \gamma, C_G; \text{Req} \)

function MakeAccurate could also rely on Property 3. It is a simple mean to obtain an accurate subsystem. The chosen configurations of observations have to guarantee some properties. The observable configurations required to make a fault $F$ diagnosable with an accurate diagnoser has to guarantee that properties obtained for diagnosing another fault $F'$ are not lost, hence the importance of the property monotony.

### 4.4 Illustrative example

We develop an example for the specification of the system illustrated in Figure 1.

We analyze the diagnosability of the fault $F1$ on the component $\Gamma_1$. The initial observable configuration is empty: $\Sigma_o = \emptyset$. We check that a solution for the sensor placement problem exists by considering all events (except the fault $F1$) as observable in the model $\Gamma_1$. Then we determine the possible observable configurations to such that $F1$ is diagnosable and $\Delta_{\Gamma_1}(F)$ is accurate. We prefer the minimal one: $\Sigma_{o_{1}} = \{e1, e2, s3\}$. The global cost $C_G^{\Gamma_1}$ includes the cost of these observations, $(C_D + C_A)$, and the monitoring cost associated to the component $\Gamma_1$, $C_M$. We now consider the subsystems of two components containing $\Gamma_1$ and another component interacting with $\Gamma_1$: $[\|\gamma_{1}\| = 1, 2, 3]$ and $[\|\gamma_{2}\| = 1, 2]$. In $\gamma_1$, $F1$ is diagnosable with an accurate diagnoser for the same observable configuration: $\Sigma_{o_{1}} = \{e1, e2, s3\}$. But the monitoring cost for two components is obviously higher than for one component. So the global cost $C_G^{\gamma_{2}}$ for this subsystem is greater than $C_G^{\Gamma_1}$. Following the same reasoning for the subsystem $\gamma_2$, we find that the global cost $C_G^{\gamma_{2}}$ is also greater than $C_G^{\Gamma_1}$. Then we prefer to consider $\Gamma_1$ only with the observable configuration $\Sigma_{o_{1}} = \{e1, e2, s3\}$ and a cost $C_G^{\Gamma_1}$. Here $\text{Req} = \{e1, e2, s3\}$ because the observable configuration was empty before starting the algorithm. We could have obtained an observable configuration guaranteeing objectives with a lower global cost by considering the subsystem $\Gamma_1$ and $\gamma_2$, if the new cost $(C_D + C_A)$ for these subsystems were smaller than the difference between the monitoring cost for component $\Gamma_1$ and the monitoring cost for $\gamma_1$ or $\gamma_2$.

Then we study the diagnosability of the fault $F2$ in the component $\Gamma_2$. The initial observable configuration is not empty anymore: $\Sigma_o = \{e1, e2, s3\}$. We first check the solution existence for the sensor placement problem by considering all events (except the fault $F2$) as observable in the model of $\Gamma_2$. There is no way to make $F2$ diagnosable in $\Gamma_2$ by techniques from the sensor placement. The only solution is to modify the structure of $\Gamma_2$.

The initial observable configuration is still $\Sigma_o = \{e1, e2, s3\}$ for the diagnosability analysis of $F3$ in $\Gamma_3$, because no more observations are brought by analysis of $F2$ (no solution for diagnosing $F2$). There exists a solution for the sensor placement problem in $\Gamma_3$, so we determine the observable configurations to make $F3$ diagnosable with an accurate diagnoser: $\{e3, e4, s4\} \in \Sigma_o$. The global cost $C_G^{\Gamma_3}$ includes the cost of the observations and the monitoring cost associated to the component $\Gamma_3$. We now consider the subsystems of two components containing $\Gamma_3$ and another component interacting with $\Gamma_3$: $[\|\gamma_{3}\| = 1, 2, 3]$. In the subsystems $\gamma_1$ and $\gamma_2$, $F3$ is diagnosable with the same observable configuration as in the component $\Gamma_3$. The monitoring cost for two components is higher than for one component, so the global costs associated to these subsystems are higher than the previous one. The fault $F3$ is diagnosable in $\Gamma_3$ with $\text{Req} = \{e3, e4, s4\}$ and a global cost $C_G^{\gamma_{2}}$.

Finally, the distributed system is diagnosable for the faults $F1$ and $F3$ with the observable configuration $\Sigma_o = \{e1, e5, e6, s2, s3, s4\}$ and a diagnostic architecture composed of two diagnosers: one on the component $\Gamma_1$ for diagnosing $F1$ and one on the component $\Gamma_3$ for diagnosing $F3$. Even if $F2$ cannot be diagnosable in the system, the diagnosis of $F2$ is accurate with the chosen observable configuration (all interactive events of $\Gamma_2$ are observable) so we could place another diagnoser on the component $\Gamma_2$.

### 5 Related Work

In the literature, there are many works about fault diagnosis and diagnosability on discrete-event systems. Our framework is based on the classical framework defined in [Sampath et al., 1995]. The problem of checking diagnosability has been studied for many years and several algorithms have been proposed for centralized monitoring architectures [Jiang et al., 2001], [Yoo and Lafortune, 2002]. Our proposal relies on the diagnosability of distributed discrete event systems that has been defined in [Pencolé, 2004]. None of these works propose any design feedback to increase the system diagnosability.

A possible way to provide design requirements for diagnosability is to solve the sensor placement problem. In the
sensor placement problem, it is assumed that there always exist modifications of type 1 (see section 3.1) by adding sensors that guarantee the diagnosability of the whole system. Moreover, the cost of the monitoring architecture is supposed to be negligible. In [Debouk et al., 1999], the problem is to select an optimal subset of available sensors by inferring a set of sensor tests. This inference assumes that the cost of \( n+1 \) sensors is always greater than the cost of \( n \) sensors \((i.e. \text{optimal} = \text{minimal})\) which may be a restrictive assumption from an economical point of view. [Jiang et al., 2003] proposes a methodology to select a minimal set of sensors to preserve properties like normality, diagnosability and proves that this problem is NP-hard. In [Narasimhan et al., 1998], the problem is defined over a qualitative model and in [Torta and Torasso, 2007] it is extended by parametrizing some discriminability relations of the system. Here also, optimality is equivalent to minimality. In [Spanache et al., 2004], the selection is performed by a genetic algorithm to make a continuous system more diagnosable. The cost of the sensor takes into account the economical issue.

To our best knowledge, there is no work about improving diagnosability in a distributed framework. In [Pencolé, 2005], the question of improving diagnosability in a distributed discrete event systems has been introduced. The idea is to detect local undiagnosable scenarios that can be used to provide design requirements for the elimination of such scenarios.

6 Conclusion and Perspectives

This paper defines a framework based on a classical model-based diagnosis formalism in order to extend automatic diagnosability analyses and provide design requirements for a distributed dynamic system. We propose to define the problem as a cost optimization problem where not only the costs about the design of the system but also the costs about the monitoring architecture are taken into account in order to minimize the integration costs of the distributed system. Finally, we claim that the design requirements for diagnosability are closely related to the design requirements for accuracy as this property is a way to isolate a subsystem above which it is possible to design a monitoring architecture that provides a diagnosis as accurate as possible. In this paper, we have presented an algorithm which selects a subsystem for which the total cost \( C_G \) is minimal. The solution is not unique but the algorithm can be extended to provide a set of possible subsystems which have a minimal cost. Our perspectives are to develop this methodology in detail by integrating the different methods (diagnosability checkers, sensor placement selectors) and automatically provide design requirements for distributed systems. Then we would like to apply this work in the Archistic project framework for the maintenance of aeronautical systems.

References

A Symbolic Approach for Component Abstraction in Model-Based Diagnosis

Gianluca Torta and Pietro Torasso
Dipartimento di Informatica, Università di Torino (Italy)

Abstract

In the present paper we address the problem of automatically synthesizing component abstractions by taking into account the level of observability of the system as well as contextual conditions within the context of the Model-Based Diagnosis task. We show how to exploit the notion of indiscriminability for abstracting the original model without losing any relevant diagnostic information. The paper presents an algorithm for the computation of abstractions that takes advantage of the symbolic compilation of the system model for giving both theoretical guarantees about the computational cost and good experimental performance on a non-trivial domain.

1 Introduction

In the Model-Based Diagnosis (MBD) field, several works have investigated the possibility of performing the diagnostic reasoning on an abstract model instead of (or before) considering a detailed model involving a larger number of components or more behavioral modes for each component (see for example [Mozetič, 1991], [Friedrich, 1993], [Autio and Reiter, 1998], [Provan, 2001], [Chittaro and Ranon, 2004]). While most of the work on abstraction for MBD has focused on the use of abstraction mappings provided by a domain expert (e.g. [Mozetič, 1991], [Provan, 2001], [Chittaro and Ranon, 2004]) some recent works have addressed the problem of automating the task of synthesizing an abstraction mapping (e.g. [Torta and Torasso, 2003], [Sachenbacher and Struss, 2005]). The work by [Sachenbacher and Struss, 2005] addresses the problem of abstracting the domains of system variables, whereas the work by [Torta and Torasso, 2003] addresses the automatic synthesis of abstract components. While component abstractions provided by users are mainly exploited for focussing diagnostic reasoning starting from the most abstract level, automatic abstraction methods mainly aim at replacing the ground model with an abstract model which still offers some guarantee of the correspondence between the abstract diagnoses and the ones at the ground level.

In the present paper we provide novel solutions to the problem of components abstraction by taking into account the level of observability of the system as well as contextual conditions. The abstraction process will stop to create new abstract components if further abstractions would loose diagnostic information or would synthesize abstract components which have too many behavioral modes. The paper provides two main contributions: on the one hand, the presented approach is more flexible than the one by [Chittaro and Ranon, 2004]; in particular it does not require that the behaviour of the system at different levels of abstractions is provided by the user, since it is automatically synthesized. Another relevant contribution is the description of an algorithm for the computation of abstraction mappings that can take advantage of the symbolic compilation of the system model for giving theoretical guarantees about the computational cost of the abstraction process.

2 Diagnosis and Indiscriminability

Before presenting our approach to abstraction, we precisely define the class of systems upon which the abstraction process can be applied and the notion of indiscriminability we use as a driving element for abstraction. We start from the definition of System Description.

Definition 2.1 A System Description SD is a pair \((SV, DT)\) where:

- \(SV\) is the set of discrete system variables partitioned in \(P\) (system ports), \(C\) (components) and \(I\) (internal endogenous variables).
- \(P\) are partitioned in \(P_{exo}\) (inputs and contexts) and \(P_{end}\) (external endogenous variables).
- \(C\) consists of \(\{C_{ok}, \ldots, C_{ok}\}\) (Domain Theory) is a set of relations (Component Domain Theories) each one modeling the behavior of a component.

As it is apparent from the definition, the description of the model is centered around components. For each component...
Given a SD, we define an observability $O_{AV}$ for SD as a subset $O_{AV} \subseteq O$; i.e. observability $O_{AV}$ specifies the endogenous variables whose values will actually be available in diagnostic problems.

Given a SD, a context restriction $X_R$ specifies the assignment of a value to each variable $x \in R \subseteq P_{exo}$. In other words, a context restriction specifies that the system will operate only in contexts $X$ (i.e. assignments to the $P_{exo}$ variables) that satisfy the constraints imposed by $X_R$.

Based on the notions of observability and context restriction, we are now able to define the indiscriminability between two different states of a subsystem $\Gamma$.

Definition 2.3 Let SD be a System Description, $O_{AV}$ be the current observability and $X_R$ be a context restriction.

Given a subsystem $\Gamma \subseteq C$, we say that two states $S_\Gamma, S'_\Gamma$ of $\Gamma$ are indiscriminable w.r.t. $O_{AV}, X_R$ iff for any instance $X$ of $P_{exo}$ variables s.t. $X_R \subseteq X$ and for each $S \Gamma$:

$$\Pi_{O_{AV}}(\sigma_{S \Gamma, X \cap S, S_R}GDT) = \Pi_{O_{AV}}(\sigma_{S'_\Gamma, X \cap S, S_R}GDT)$$

where $S \wedge X$ is a status of the subsystem $\sim \Gamma = C \setminus \Gamma$ complementary to $\Gamma$.

According to the above definition, two states $S_\Gamma, S'_\Gamma$ of the subsystem $\Gamma$ are considered indiscriminable when there is no observable difference between $S_\Gamma, S'_\Gamma$ in all of the situations s.t. the observability is $O_{AV}$ and the context satisfies $X_R$. While the notion of indiscriminability strictly depends on the specification of an observability $O_{AV}$ (this specifies which is the observational trace), the use of a context restriction $X_R$ is just a possibility for performing more accurate analysis of indiscriminability when the user knows in advance under which contextual situation the system is working. If such knowledge is missing, context restriction $X_R$ is put to True and therefore all possible combinations of contexts and commands are considered in the analysis. Note that the indiscriminability relation induces a partition into indiscriminability classes of the set of possible behavioral modes of $c$, i.e. of $D(c)$.

There is an obvious relation between the diagnoses of a diagnostic problem and the notion of indiscriminability. Let us consider a diagnostic problem $DP = (SD, X, Y)$ where $Y$ is defined in terms of an observability $O_{AV}$ and $X$ respects the constraints imposed a context restriction $X_R$. If we have that $S_\Gamma, S'_\Gamma$ of $\Gamma$ are indiscriminable w.r.t. $O_{AV}, X_R$ and $D = S_\Gamma \cup S'_\Gamma$ is a diagnosis for the diagnostic problem DP, then $D' = S'_\Gamma \cup S_{\Gamma}$ is an alternative diagnosis for DP.

More states of a subsystem are discovered indiscriminable, larger is the set of diagnoses. The basic idea of our approach is to exploit the indiscriminability for guiding the process of abstraction as it will be discussed in the next section.

3 Abstraction Mappings

Since we are interested in abstracting a subsystem into an abstract component, we introduce the notion of an abstraction mapping for specifying not only which components are merged into an abstract component $AC_\Gamma$ , but also how the behavior of the components of the subsystem $\Gamma$ can be abstracted into the behavior of $AC_\Gamma$ which is intended to replace the subsystem $\Gamma$.
Definition 3.1 An Abstraction Mapping $AM\Gamma$ maps a subsystem $\Gamma = \{c_1, \ldots, c_m\}$ to an abstract component $AC\Gamma$ with behavioral modes $\{abm_1, \ldots, abm_k\}$ s.t.:
- $c_1, \ldots, c_m$ are the subcomponents of $AC\Gamma$.
- $P_{end}(AC\Gamma) = P_{end}(\Gamma)$ and $P_{exo}(AC\Gamma) = P_{exo}(\Gamma)$.
- Each subcomponent $abm_i \in \text{DOM}(AC\Gamma)$ is associated with a non-empty subset of states of $\Gamma$ (i.e., instances of components $c_1, \ldots, c_m$) s.t. any possible status $S\Gamma$ is associated with exactly one abstract behavioral mode $abm_i$.

From the above definition it is apparent that the exogenous and the endogenous ports of the abstract component $AC\Gamma$ are directly derived from the ones of the subsystem $\Gamma$, while the endogenous ports of $c_1, \ldots, c_m$ which connect just components inside $\Gamma$ are forgotten in abstract component $AC\Gamma$. This represents an actual simplification of the model of the abstract component.

As concerns the abstract behavioral modes $abm_i \in \text{DOM}(AC\Gamma)$ there is no obvious requirement that each abstract behavioral mode corresponds to at least one status of $\Gamma$. More interestingly, the definition imposes the requirement that the abstraction mapping is mutually exclusive and complete, that is each status of $\Gamma$ has a counterpart in an abstract behavioral mode and no no status of $\Gamma$ is associated to more than one abstract behavioral mode. In the above definition there is no requirement on the (in)discriminability of the abstract behavioral modes of the abstract components with respect to a given observability $O_{AV}$. The following definition imposes such a requirement by taking into account the goal of exploiting indiscriminability for driving the abstraction process.

Definition 3.2 An Abstraction Mapping $AM\Gamma$ is correct w.r.t. the degree of observability $O_{AV}$ and a context restriction $X_R$ if $O_{AV} \cap I(\Gamma) = \emptyset$ and all the states associated by $AM\Gamma$ with an abstract behavioral mode $abm_i$ are pairwise indiscriminable w.r.t. $O_{AV}$, $X_R$.

Condition $O_{AV} \cap I(\Gamma) = \emptyset$ requires that none of the currently observed variables $O_{AV}$ is an internal variable of $\Gamma$ that would disappear in the abstraction. Moreover, if the Abstraction Mapping associates $S\Gamma_i$, $S\Gamma_j$ to the same abstract behavioral mode, we require that $S\Gamma_i$, $S\Gamma_j$ are indiscriminable w.r.t. $O_{AV}$ and $X_R$ according to Definition 2.3. In this way, only states of $\Gamma$ that are indiscriminable w.r.t. to a given level of observability $O_{AV}$ and a contextual restriction $X_R$ can be abstracted into the same abstract behavioral mode of the abstract component $AM\Gamma$.

Note that an Abstraction Mapping can be interpreted as an equivalence relation on the states of $\Gamma$ s.t. each equivalence class corresponds to an abstract behavioral mode of $AC\Gamma$.

The notion of abstraction mapping for a subsystem can be extended to the whole systems. In particular, a System Abstraction Mapping $AM$ is a set of Abstraction Mappings $AM\Gamma_1, \ldots, AM\Gamma_p$, s.t. $\Gamma_1, \ldots, \Gamma_p$ form a partition of the system components $C$.

Given an abstraction mapping, we can apply it to a System Description in order to get an abstract SD.

Definition 3.3 Let $SD = (SV, DT)$ be a System Description and $AM = \{AM\Gamma_1, \ldots, AM\Gamma_p\}$ be a System Abstraction Mapping. We define the abstraction $SD_A = (SV_A, DT_A)$ of $SD$ as follows:
- $C_A = \{AC\Gamma_1, \ldots, AC\Gamma_p\}$
- $P_{A,exo} = P_{exo} P_{A,end} = P_{end}$
- $I_A = \left( \bigcup_{i=1}^{p} P_{end}(\Gamma_i) \right) \setminus P_{A,end}$
- for each $AC\Gamma_i$, let $V_i = (V_{\Gamma_i} \cup P_{exo}(\Gamma_i) \cup P_{end}(\Gamma_i))$ if $DT(\Gamma_i)$ is obtained by replacing in each tuple of $V_i$ (of $DT(\Gamma_i)$) the assignment $S\Gamma_i$ to the $\Gamma_i$ variables with the assignment $abm_i$ to $AC\Gamma_i$ iff $S\Gamma_i$ is mapped to $abm_i$ according to $AM$.

The abstract system description $SD_A$ involves as its components the abstract components $AC\Gamma_1, \ldots, AC\Gamma_p$ specified in the system abstraction mapping. The endogenous and exogenous ports of the abstract system are the same as the ones of the ground system (i.e. $P_{A,exo} = P_{exo}, P_{A,end} = P_{end}$), while the internal variables $I_A$ are the union of the endogenous variables of all the subsystems $\Gamma_1, \ldots, \Gamma_p$ involved in the system description (obviously the variables which are the system endogenous ports are not included in the set of internal variables).

As concerns the domain theory of each abstract component $AC\Gamma_i$, this can be obtained from the $GDT(\Gamma_i)$ of the subsystem $\Gamma_i$. In particular, for each tuple of $GDT(\Gamma_i)$ of the form $(\alpha, S\Gamma_i)$ (where $\alpha$ is an assignment to the exogenous and endogenous variables of $\Gamma_i$ and $S\Gamma_i$ is a status of $\Gamma_i$) we add a tuple $(\alpha, abm_i)$ to $DT(AC\Gamma_i)$ if the abstraction mapping prescribes that the status $S\Gamma_i$ has to be mapped into the abstract behavioral mode $abm_i$. Moreover, after these transformations, the internal variables of subsystem $\Gamma_i$ must be removed from $DT(AC\Gamma_i)$ (with a project on $V_i$ variables) because abstract components (as ground components) do not have internal variables.

It is worth noting that the notion of correct abstraction mapping provides an important guarantee as concerns the relation between the result of diagnostic problem solving performed at the ground and abstract levels.

Let $DP = (SD, X, Y)$ be a diagnostic problem at the ground level and let $DP_{AM} = (SD_A, X, Y)$ be the corresponding abstract diagnostic problem obtained via the system abstraction mapping $AM = \{AM\Gamma_1, \ldots, AM\Gamma_p\}$. It is worth noting that the commands $X$ and the observations $Y$ are exactly the same at the ground level and at abstract level.

If $GBM$ is an assignment of a behavioral mode to each component $c \in C$ we denote with $ABM$ the assignment of an abstract behavioral mode to each abstract component $\{AC\Gamma_1, \ldots, AC\Gamma_p\}$ according the abstraction mapping $AM = \{AM\Gamma_1, \ldots, AM\Gamma_p\}$. In particular, if $GBM = S\Gamma_1, \ldots, S\Gamma_p$ (where $S\Gamma_i$ is a status of $\Gamma_i$) and the system abstraction mapping $AM$ prescribes that the status $S\Gamma_i$ has to be mapped into the abstract behavioral mode $abm_i$, then $ABM = abm_1 \cup \ldots \cup abm_p$ is the abstract correspondent of $GBM$. In other words, the system abstraction mapping $AM$ maps a status of the system at the ground level into an abstract status of the abstract system. It is also worth noting that the inverse $AM^{-1}$ of the abstraction mapping $AM$ can be used for mapping an abstract status of the abstract system into a set of states at the ground level.
The possibility of performing diagnostic reasoning at abstract level instead at the ground level without losing any diagnostic information is proved by the following property.

**Property 3.1** Let $AM$ be a correct System Abstraction Mapping w.r.t. observability $O_{AV}$ and context restriction $X_R$. Moreover, let $SD_A$ be the abstraction of $SD$ according to system Abstraction Mapping $AM$ and let $DP = (SD, X, Y)$ be a diagnostic problem s.t. $Y$ is an assignment of $O_{AV}$ variables and $X$ satisfies the constraints imposed by $X_R$. Let $DP_{AM} = (SD_A, X, Y)$ be the abstract diagnostic problem corresponding to $DP$ according to $AM$.

For any ground status $GBM$ that is a diagnosis for the diagnostic problem $DP$, the $ABM$ obtained by $GBM$ via $AM$ is an abstract diagnosis for $DP_{AM}$. Moreover, for any $ABM$ that is an abstract diagnosis for $DP_{AM}$, all the ground states $GBM$ that can be obtained by $ABM$ via the inverse abstraction mapping $AM^{-1}$ are diagnoses for $DP$.

The above property states that each diagnosis for a specific diagnostic problem $DP$ at the ground level has a corresponding abstract diagnosis for the same diagnostic problem solved at abstract level $DP_{AM}$. This property (called downward-failure property in [Ott et al., 1994]) guarantees that an assignment that is consistent at the ground level (i.e., a ground diagnosis) is necessarily mapped to an assignment that is consistent at the abstract level (i.e., an abstract diagnosis). However, the property tells much more: we can map back any abstract diagnosis to its corresponding set of diagnoses at the ground level by using the inverse abstraction mapping $AM^{-1}$ of the abstraction mapping $AM$ (this corresponds to the ID property introduced in [Ott et al., 1994]). The above property implies that there is no loss of diagnoses if we solve a diagnostic problem at the abstract level instead of solving it at the ground level. More specifically, we get the same set of ground diagnoses if we solve the diagnostic problem at the ground level or we first apply the abstraction mapping to the ground model to get an abstract model and then we solve the diagnostic problem at the abstract level and finally we map back the abstract diagnoses into the corresponding set of ground diagnoses.

It is worth noting that this result is not obvious. As shown by [Autio and Reiter, 1998] the correspondence between abstract and ground diagnoses is sometimes troublesome: in particular if one chooses to define the abnormality of the abstract component in terms of the abnormality of at least one the ground components, some diagnoses at the ground level can be lost. This is due to the fact that such abstractions can map indiscriminable instances of ground behavioral modes to different abstract modes (in particular, $ok$ and $ab$). This does not happen in our approach since we have introduced the notion of correct system abstraction mapping based on indiscriminability.

### 4 An Example System

In order to illustrate the main concepts introduced in the paper, we will use as an example through the paper the hydraulic system reported at the top of Figure 1. It is worth noting that the components and the topology of the hydraulic system are exactly the ones used in [Chittaro and Ranon, 2004].

![Figure 1: The Hydraulic System and its Abstractions.](image-url)

Figure 2 reports the domain models of a generic pipe, pump, valve, join and split. These models are expressed in terms of qualitative equations involving both qualitative deviations [Struss et al., 1996] and qualitative (absolute) values. As usual, a qualitative deviation can take value in the set $\{-0, +\}$, while the possible qualitative values are $\{0, +\}$ for the flows and $\{+, \infty\}$ for the resistances.

For example, when a pipe is in the leaking mode $(lk)$, the absolute value $f_{out}$ of the flow at the end of the pipe is the same as the flow $f_{in}$ at the beginning of the pipe. However, the qualitative deviation $\Delta f_{out}$ is $\Delta f_{in} + +$, i.e. the deviation of the flow is qualitatively decreased at one end of the pipe with respect to the other end.

It is worth noting that the qualitative equations used for capturing the behavior of the components require the adoption of a relational representation, since in most cases a qualitative equation involves non determinism. The actual formalism adopted for modeling the components and the system is a relational form: we have used qualitative equations just because they are more compact and easier to understand. It is straightforward to translate the domain model expressed via qualitative equations into relational form by taking into account the simple semantics of sign algebra.

The hydraulic system has three exogenous commands: the expected operating conditions $oc_V1$ and $oc_V2$ of the two valves (each valve could be open or closed) and the qualitative deviation $\Delta s_{Pump}$ of the control signal of $Pump$ (that influence the quantity of fluid pumped by $Pump$). Therefore: $P_{exe} = \{\Delta s_{Pump}, oc_V1, oc_V2\}$

As concerns observability, the maximum degree of observability involves the ports marked in figure 1, e.g. $PUMP_{out}$ which is the connection point between $Pump$ and $P1$. For each connecting point we have up to four observable variables (the qualitative absolute values of flow and resistance,
as well as the qualitative deviations of them). So the maximum degree of observability involves 32 variables. 

\[ O = \{ \Delta f_{\text{Pumpout}}, \Delta r_{\text{Pumpout}}, f_{\text{Pumpout}}, r_{\text{Pumpout}} \} \]

\[ \Delta f_{\text{split}} \Delta r_{\text{split}}, f_{\text{split}}, r_{\text{split}} \]

\[ \Delta f_{\text{P2out}}, \Delta r_{\text{P2out}}, f_{\text{P2out}}, r_{\text{P2out}} \]

\[ \Delta f_{\text{V1out}}, \Delta r_{\text{V1out}}, f_{\text{V1out}}, r_{\text{V1out}} \]

\[ \Delta f_{\text{P3out}}, \Delta r_{\text{P3out}}, f_{\text{P3out}}, r_{\text{P3out}} \]

\[ \Delta f_{\text{V2out}}, \Delta r_{\text{V2out}}, f_{\text{V2out}}, r_{\text{V2out}} \]

\[ \Delta f_{\text{Joinout}}, \Delta r_{\text{Joinout}}, f_{\text{Joinout}}, r_{\text{Joinout}} \]

\[ \Delta f_{\text{P0out}}, \Delta r_{\text{P0out}}, f_{\text{P0out}}, r_{\text{P0out}} \]

Let us now illustrate the notion of indiscernibility w.r.t. to a given level of observability and contextual restriction, by considering the subsystem \( \Gamma = \{ P2, V1 \} \) involving the pipe P2 and the valve V1.

In case the observability \( O_{AV}^1 = O \{ \Delta f_{\text{P2out}}, \Delta r_{\text{P2out}}, f_{\text{P2out}}, r_{\text{P2out}} \} \) (i.e. all the observable variables are observed but the endogenous ports connecting P2 with V2) and the context restriction \( X_R \) is TRUE (that is there is no contextual restriction), we have that two states \( S_1 = \{ P2(c1), V1(ok) \} \) and \( S_2 = \{ P2(ok), V1(sc) \} \) are indiscernible according to definition 2.3 (\( c1 \) stands for clogged and \( sc \) for stuck-closed).

Under the same assumptions, we have that \( \{ P2(ok), V1(OK) \} \) and \( \{ P2(ok), V1(so) \} \) are indiscernible, whereas if we have no trivial contextual restriction \( X_R = oc \cdot V1(\text{closed}) \) (i.e. we know that the valve V1 will be in the operating condition open) \( \{ P2(ok), V1(OK) \} \) and \( \{ P2(ok), V1(so) \} \) become indiscernible.

As second example, let us consider a case where \( O_{AV}^1 = O_{AV}^1 \) and the \( X_R = \{ \Delta s = + \} \) (i.e. the control signal of the Pump requires an increase in pumping).

The following Abstraction Mapping \( AM_{P2,V1} \) maps the subsystem \( \{ P2, V1 \} \) into the abstract component \( P2V1 \) with 5 behavioral modes.

\[ abm_1 = \{ (ok, ok) \}, abm_2 = \{ (ok, so) \} \]

\[ abm_4 = \{ (lk, ok) \}, abm_5 = \{ (lk, so) \} \]

As concerns the abstract component \( P2V1, P_{exo}(P2V1) \) contains just the operating condition of valve V1 (i.e. oc \cdot V1) whereas \( P_{end}(P2V1) = \{ \Delta f_{\text{P2in}}, \Delta r_{\text{P2in}}, f_{\text{P2in}}, r_{\text{P2in}}, \Delta f_{\text{V1out}}, \Delta r_{\text{V1out}}, f_{\text{V1out}}, r_{\text{V1out}} \} \).

The Abstraction Mapping \( AM_{P2,V1} \) specifies that ground status \( (ok, ok) \) have to be mapped in the abstract behavioral mode \( amb_1 \), whereas the states \( (ok, sc), (lk, sc), (cl, sc), (cl, ok), (cl, so) \) have to be mapped into \( amb_3 \), and so on. It is easy to see that the 5 abstract modes form a partition of the possible states of the subsystem \( P2V1 \). More interesting, the Abstraction Mapping \( AM_{P2,V1} \) reported above is correct w.r.t to observability \( O_{AV}^1 \) and the contextual restriction \( X_R = \{ \Delta s = + \} \). This also implies that all the states listed in \( abm_5 \) are pairwise indiscernible by taking into consideration the definition 2.3. Since the abstraction mapping is correct we have the guarantee that no useful diagnostic information is lost if we use the abstract component \( P2V1 \) instead of the ground model of P2 and V1 in the global model of the hydraulic system. Obviously this guarantee is valid as far as the diagnostic problems satisfy observability \( O_{AV}^1 \) and Contextual restriction \( \Delta s = + \).

If we consider observability \( O_{AV}^1 = \{ \Delta f_{\text{P2out}}, \Delta r_{\text{P2out}}, f_{\text{P2out}}, r_{\text{P2out}} \} \) (which involves just the system endogenous ports), it is easy to see that the abstraction mapping \( AM_{P2,V1} \) with behavioral modes \( abm_1', abm_2', abm_3' \) s.t.: 

\[ abm_1' = \{ (ok, ok), (lk, ok) \}, abm_2' = \{ (ok, so), (lk, so) \} \]

\[ abm_3' = \{ (ok, sc), (lk, sc), (cl, sc), (cl, ok), (cl, so) \} \]

is correct with respect observability \( O_{AV}^1 \) and context restriction \( \Delta s = + \). Some of the states of the subsystem \( P2, V1 \) that are discriminable under observability \( O_{AV}^1 \) are no more discriminable with observability \( O_{AV}^2 \) and therefore can be abstracted in the same abstract behavioral mode. For example \( (ok, ok) \) and \( (lk, ok) \) are abstracted in \( abm_1' \): this shows that the abstraction process based on indiscernibility is able to recognize that at a given level of observability the normality of a subsystem (i.e. \( (ok, ok) \)) status has to be merged with some of the faulty states (e.g. \( (lk, ok) \)).

This example shows also that our characterization of abstraction takes into consideration the reduction of observability not only when it directly impacts the ports of the subsystem to be abstracted but also a reduction of observability in the whole system.

5 Computing Mappings and Abstractions

In this section we present algorithms for the automatic synthesis of a correct Abstraction Mapping \( AM \) given a System Description \( SD \), an observability \( O_{AV} \) and a context restriction \( X_R \). One of the computational challenges of this task is the complexity of the search for the subsystems \( \Gamma \) of ground components to be mapped to abstract components \( AC \) and of the representation of the corresponding mappings \( AM_{\Gamma} \). We discuss our approach to deal with this problem later in this section.

Another difficulty concerns the manipulation of the Global Domain Theory \( GDT \) . Indeed, in order to check whether
Compute $\text{DTo}(o, \text{sw}_o, \text{rd}_o)$
\begin{align*}
1 & D(s_w_v) = \{\text{yes, no}\}, D(r_d_v) = D(o) \cup \{\text{abs}\} \\
2 & D_T_o = \emptyset \\
3 & \text{foreach } v \in D(o) \\
4 & D_T_o = D_T_o \cup \{(o(v), s_w_v(yes), r_d_v(v))\} \\
5 & D_T_o = D_T_o \cup \{(o(v), s_w_v(no), r_d_v(abs))\}
\end{align*}

Compute $\text{XSD}(SD = (SV, DT))$
\begin{align*}
1 & \chi_{SV} = SV, X_DT = DT \\
2 & SW_0 = \emptyset, RD_0 = \emptyset \\
3 & \text{foreach } o \in O \\
4 & SW_0 = SW_0 \cup \{sw_o\}, RD_0 = RD_0 \cup \{rd_o\} \\
5 & \text{ComputedTo}(o, sw_o, rd_o) \\
6 & X_DT = X_DT \cup \{DT_o\} \\
7 & \chi_{SV} = \chi_{SV} \cup SW_0 \cup DT_0
\end{align*}

Figure 3: Computing the Extended System Description.

two states $S_T, S_F$ of a subsystem $\Gamma$ are indistinguishable w.r.t. $O_{AV}, X_R$ (and therefore can be correctly merged), we must check the equality of the relations $\Pi_{O_{AV}}(\sigma_{S_T}, X_R, GDT)$ and $\Pi_{O_{AV}}(\sigma_{S_F}, X_R, GDT)$ (as a direct consequence of Definition 2.3).

Since $GDT$ can be a huge relation, such a check could be very expensive from a computational point of view. In order to improve on this situation, we adopt a symbolic representation of the relations (among which, $GDT$) as OBDDs (Ordered Binary Decision Diagrams). OBDDs have been used to efficiently store and manipulate huge relations that would have been impossible to represent extensionally; however, in order to have a theoretical guarantee that the check for indistinguishability is efficient given an efficient encoding of the Global Domain Theory, we first need to slightly extend the original model $SD$.

In particular, we need to introduce new variables for explicitly representing within the model the possibility of having different degrees of observability. More precisely we introduce two new variables for each observable variable according to the following definition.

**Definition 5.1** Given an observable variable $o \in O$ its associated switch $sw_o$ is a variable with $D(sw_o) = \{\text{yes, no}\}$. The reading $rd_o$ of $o$ is a variable s.t. $D(rd_o) = D(o) \cup \{\text{abs}\}$.

The switch $sw_o$ associated with $o$ is intended to specify whether $o$ is actually observed or not. The reading $rd_o$ of $o$ is intended to represent an endogenous variable whose value is the same as the value of $o$ if $sw_o(yes)$, and $abs$ (absent) otherwise.

The computation of the Global Domain Theory of the extended model (denoted $XGDT$) is made in two steps. First, we extend $SD$ into an Extended System Description $XSD$; then, we compute $XGDT$ from $XSD$.

Function $\text{ComputeXSD}$ (Figure 3) builds $XSD$ by augmenting $SD$ with switch variables, reading variables and new relations $DT_o$ that enforce their intended meaning (function ComputeTo).

Once $XSD$ has been computed, relation $XGDT$ can be obtained in a similar way as $GDT$ is obtained from $SD$. In particular, we join the relations that compose $XDT$ and for get the variables that are useless in the global model:
\[ XGDT = \Pi_{XVARS} DT_{\tau_1} \times \ldots \times DT_{\tau_m} \times DT_{\text{sw}} \ldots DT_{\text{sw}} \]
where $XVARS = \cup_{v \in O} SW_v \cup RD_0$. Note that variables $O$ are forgotten from $XGDT$. The role of such variables is played in $XGDT$ by their readings $RD_0$ whose values, contrary to the values of $O$, are suitably influenced by the values of the switches $SW_0$.

Function ComputeMapping (Figure 4) computes Abstraction Mappings $\lambda_{AM}$ in a hierarchical form, where two (abstract) components at a time are merged.

**Definition 5.2** A Hierarchical Abstraction Mapping $\lambda_{AM_{i,j}}$ is a tuple $(AC_{i,j}, H\lambda_{AM_{i,j}}, C_{\lambda_{AM_{i,j}}}, \tau_{i,j})$ where $AC_{i,j}$ is the name of an abstract component; $H\lambda_{AM_{i,j}}, H\lambda_{AM_{j,i}}$ are Hierarchical Abstraction Mappings which define the two (abstract) subcomponents $AC_i, AC_j$; and $\tau_{i,j}$ is a set of tuples $(abm, \lambda)$ representing the behavioral modes of $AC_{i,j}$. In particular, $(abm, \lambda)$ defines behavioral mode $abm$ of $AC_{i,j}$ to the set $\lambda$ of pairs $(abm_{\tau_1}, abm_{\tau_2})$ s.t. $abm_{\tau_1}$ is a behavioral mode of $AC_i$ and $abm_{\tau_2}$ is a behavioral mode of $AC_j$.

At the bottom of a Hierarchical Abstraction Map there are special mappings $\lambda_{AM_{i,j}}$ that wrap ground components $c$ by defining an abstract component $AC_c$ whose behavioral modes $abm$ are mapped to subsets of indiscriminably behavioral modes of $c$. Mapping $\lambda_{AM_{i,j}}$ can then be viewed as a way to compactly represent an Abstraction Mapping $\lambda_{AM_{i,j}}$, s.t. the subsystem $\Gamma_{i,j}$ mapped by $\lambda_{AM_{i,j}}$ is the set of ground components that appear at the leaves of $\lambda_{AM_{i,j}}$.

Function ComputeMapping takes an Extended Global Domain Theory $XGDT$, a parameter $O_{AV}$ expressing a degree of observability $O_{AV}$ as a set of assignments to the switches $SW_0$ and a context restriction $X_R$. First of all, it restricts $XGDT$ with the observability $O_{AV}$ and the context restriction $X_R$, and then forgets the $SW_0$ variables (which are no longer needed since all of the tuples in $XGDT$ now agree on their value $O_{AV}$). In order to bootstrap the abstraction, mapping $\lambda_{AM}$ is initialized with the set of wrappers (see above) of all the ground components $c \in C$ (function LeafMapping, not reported due to lack of space).

The main loop of ComputeMapping chooses nondeterministically two of the mappings $\lambda_{AM_{i,j}}$ and $\lambda_{AM_{j,i}}$ from the set of mappings currently contained in $\lambda_{AM}$; at some point, choose will not return a pair of mappings but a special value (e.g. $\emptyset$) meaning that the algorithm should no longer attempt to perform further abstractions and should return the mapping computed so far.

Within the body of the while loop, function Merge merges $\lambda_{AM_{i,j}}$ and $\lambda_{AM_{j,i}}$ into a single mapping $\lambda_{AM_{i,j}}$ which defines a new abstract component $AC_{i,j}$. The resulting mapping is tested to check whether it is a good abstraction (function isGood); if the check is positive, mappings $\lambda_{AM_{i,j}}$, $\lambda_{AM_{j,i}}$ are replaced by $\lambda_{AM_{i,j}}$ in $\lambda_{AM}$.

What is a good abstraction? In this paper, we take the natural answer that it is a mapping $\lambda_{AM_{i,j}}$ which defines an abstract component $AC_{i,j}$ which has a small number of behavioral modes compared to the number of states of the subsystem $\Gamma_{i,j}$ of ground components abstracted in $AC_{i,j}$. Since in the worst case the number $|D(AC_{i,j})|$ of such behavioral modes
ComputeMapping$(XGDT, O_{AV}, \mathcal{X}_R)$

1. $\text{VARS} = C \cup P_{\text{exe}} \cup RDO$
2. $XGDT = \sigma_{\text{NA}} XGDT$
3. $XGDT = \sigma_{\text{NA}} XGDT$
4. $XGDT = \text{IVARS} XGDT$
5. $\mathcal{H}M = \{\text{LeafMapping}(XGDT, c) : c \in C\}$
6. while (choose($\mathcal{H}M$) = ($\mathcal{H}M_1$, $\mathcal{H}M_2$))
7. $\mathcal{H}M_{i,j} = \text{Merge}(XGDT, \mathcal{H}M_1, \mathcal{H}M_2)$
8. if (is-good($\mathcal{H}M_1, \mathcal{H}M_2$))
9. $\mathcal{H}M = \mathcal{H}M \cup \{\mathcal{H}M_{i,j}\}$
10. return $\mathcal{H}M$

Merge($XGDT, \mathcal{H}M_1, \mathcal{H}M_2$)

1. $\tau_{i,j} = \emptyset$
2. $\lambda_{i,j} = \{(abm_k, abm_m)\}$
3. $(abm_k \in D(AC_i), \text{abm}_m \in D(AC_j))$
4. foreach $(abm_k, \text{abm}_m) \text{ s.t. :}$
5. $\lambda_{i,j} \neq \emptyset$, $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
6. foreach $(abm_k, \text{abm}_m) \neq (abm_k, \text{abm}_m) \text{ s.t. :}$
7. $\lambda_{i,j} \neq \emptyset$, $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
8. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
9. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
10. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
11. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
12. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
13. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
14. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
15. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
16. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
17. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
18. $\text{abm}_q \in D(AC_i), \text{abm}_q \in D(AC_j)$
19. $\mathcal{H}M_{i,j} = \{\text{LeafMapping}(\mathcal{H}M_1, \mathcal{H}M_2), \tau_{i,j}\}$
20. return $\mathcal{H}M_{i,j}$

Figure 4: Implementation of ComputeMapping.

is $\prod_{c \in \Gamma_1 \cup \Gamma_2} |D(c)|$ (i.e. exponential in $|\Gamma_1 \cup \Gamma_2|$), it is reasonable to accept $\Gamma_{1,j}$ if $|D(\Gamma_{1,j})| \leq \sum_{c \in \Gamma_1} \alpha_c \cdot |D(c)|$ (i.e. it is linear in $|\Gamma_1 \cup \Gamma_2|$); in particular, if we let $\alpha_c = 1$ for all $c$, we can require that $|D(\Gamma_{1,j})| \leq \sum_{c \in \Gamma_1} |D(c)|$.

Due to lack of space, we cannot give a detailed comment of $\mathcal{MERGE}$ (which is reported in Figure 4). We just note that every pair $(\text{abm}_k, \text{abm}_m)$ of behavioral modes ($\text{abm}_k$ of $AC_i$, $\text{abm}_m$ of $AC_j$) is checked for indistinguishability w.r.t. any other pair $(\text{abm}_k, \text{abm}_m)$. The check is performed by getting, for each of the four abstract behavioral modes, a ground representative; in particular, $\text{abm}_k$ and $\text{abm}_m$ are arbitrary states of the subsystem $\Gamma_1$ (abstracted by $AC_i$) which map to $\text{abm}_k$ and $\text{abm}_m$, respectively; similarly, $\text{abm}_k$ and $\text{abm}_m$ are arbitrary states of the subsystem $\Gamma_1$ (abstracted by $AC_j$) which map to $\text{abm}_k$ and $\text{abm}_m$, respectively. The union $\text{abm}_k \cup \text{abm}_m$ is then checked against the union $\text{abm}_k \cup \text{abm}_m$ of the representatives of $\text{abm}_k$, $\text{abm}_m$ for indistinguishability in lines 13-15: is they turn out to be indistinguishable, $(\text{abm}_k, \text{abm}_m)$ is put in the same class as $(\text{abm}_k, \text{abm}_m)$: such a class will eventually become the definition of a new abstract behavioral mode $\text{abm}_k \cup \text{abm}_m$ of $AC_i$.

An OBBDD-based implementation of ComputeMapping receives an OBBDD $O(XGDT)$ which encodes $XGDT$ and performs all the relational operations through suitable OBBDD operations. The complexity of such an implementation is stated in the following properties. The first property concerns the complexity of Merge, which is addressed by the second property.

**Property 5.1** An OBBDD based implementation of function ComputeMapping takes time linear in the size of $O(XGDT)$. Moreover, the while loop is performed at most $n' \cdot (n' - 1)$ times where $n'$ is the cardinality of the set of ground components $C' \subseteq C$ that can be selected by choose.

First of all, ComputeMapping takes linear time in the OBBDD encoding of $XGDT$; moreover, it takes quadratic time in the number of ground components that can be possibly considered by choose. Recall that two (abstract) components can be considered for merging only if such a merging does not cause any observable in $O_{AV}$ to be removed from the model; therefore, parameter $n'$ in the property may be much smaller than the number $|C'|$ of ground components.

**Property 5.2** An OBBDD based implementation of function Merge performs at most $((2^{|D_{\text{max}}|})^2 - 1)/2$ executions of the inner loop, where $D_{\text{max}}$ is the maximum between the domains of $AC_i$ and $AC_j$. The body of the inner loop takes time linear in the size of $O(XGDT)$.

This property further justifies our aim at only building abstract components with a limited number of behavioral modes, since in such a way $D_{\text{max}}$ is not allowed to grow too much. A notable result which strongly exploits the properties of the OBBDD encoding is the fact that the inner body (and, in particular, the check for equality of $XGDT_{k,l}$ and $XGDT_{p,q}$) is again linear in the OBBDD encoding of $XGDT$.

### 6 Experimental Results

The approach presented in the paper has been tested on the hydraulic system introduced in section 4. The experiments have been conducted with a Java implementation of the algorithms that uses the JBBDD interface to the Buddy C++ library for the OBBDD operations. The test machine was equipped with an Intel Core Duo CPU at 2.4GHz with 2GB of RAM.

The OBBDD encoding of the $GDT$ involves 12839 nodes, whereas the OBBDD encoding the resulting $XGDT$ has a size of 55794 nodes. During the computation of $XS_D$, a switch and a reading are added for each observable variable $o \in O$, so the total number of additional variables is 64.

Let us first consider the subsystem composed by $\{P_2, V_1, P_4\}$ and an observability $O_{AV}^4$ which does not contain variables of the subsystem:

$$O_{AV}^4 = \{\Delta f_{\text{Pump}_{\text{out}}}, \Delta f_{\text{Pump}_{\text{out}}}, f_{\text{Pump}_{\text{out}}}, f_{\text{Pump}_{\text{out}}} \Delta f_{\text{Split}_{\text{in}}}, \Delta f_{\text{Split}_{\text{in}}, f_{\text{Split}_{\text{in}}}, f_{\text{Split}_{\text{in}}} \Delta f_{\text{Join}_{\text{out}}}, \Delta f_{\text{Join}_{\text{out}}, f_{\text{Join}_{\text{out}}} \Delta f_{P6_{\text{out}}}, \Delta f_{P6_{\text{out}}}, f_{P6_{\text{out}}}, f_{P6_{\text{out}}} \}$$

Let us also suppose that $X_R$ is TRUE (no contextual restriction). The abstraction mapping computed for the abstract component $P_2\cup V_1 \cup P_4$ has just 5 behavioral modes: $\text{abm}_1 = \{(ok, ok, ok)\}$, $\text{abm}_2 = \{(ok, ok, lk), (lk, ok, ok), (lk, ok, lk)\}$, $\text{abm}_3 = \{(ok, so, ok)\}$, $\text{abm}_4 = \{(ok, so, lk), (lk, so, lk)\}$ and $\text{abm}_5$ (all the remaining 19
ground states). It is worth noting that the number of abstract modes is small w.r.t to the total possible number (27) and is well below the threshold (9) required by is-good.

If we consider subsystem \{P3, V2, P5\}, observability \(O_{AV}^3\) and \(x_R = ocv_2\text{(closed)}\) the algorithm builds an abstraction mapping for \(P_3, V_2, P_5\) which involves just three abstract behavioral modes \(abm_1 = \{(ok, so, ok)\}, abm_2 = \{(ok, so, lk), (lk, so, ok), (lk, ok, lk)\} and \(abm_3\) (all the remaining 23 ground states). Figure 1 (central portion) reports the hydraulic system after abstract components \(P_2, V_1, P_4\) and \(P_3, V_2, P_5\) have replaced their corresponding subsystems.

As a final example let us consider the large subsystem involving Split, P2, V1, P4, P3, V2, P5 and Join. Under observability \(O_{AV}^4\) and \(x_R = ocv_2\text{(closed)}\), ComputeMapping is able to build an abstraction mapping for such a large subsystem. The most important result is the fact that component \(SI\) has just six abstract behavioral modes. The ability of our methodology for automatically synthesizing a good abstract component is evident when we compare the 6 abstract behavioral modes with the 36 potential ones. Note that observability \(O_{AV}^4\) prevents us from attempting further abstractions. The resulting system is reported at the bottom of Figure 1.

Finally, the CPU time for computing the abstraction mapping for \(SI\) is very low (less than 100 msec).

7 Discussion and Conclusions

In the present paper we have addressed the problem of automatically computing abstraction mappings which specify how a set of components can be abstracted into an abstract component. The final goal is to perform diagnostic reasoning on the abstract model instead of the ground model without losing diagnostic information. The abstraction process is driven by the indiscriminability among the observational traces of the behavior of a set of components and is strongly influenced by the level of observability.

While [Sachenbacher and Struss, 2005] have shown how to perform automatic abstraction of qualitative variable domains by taking into consideration observability (as well as the granularity of the desired abstractions), the work by [Chittaro and Ranon, 2004] has addressed the problem of adapting a hierarchical abstraction in terms of components/subcomponents to a specific level of observability. The goal of the present paper is quite similar to the one of that paper, with a very significant difference. While in [Chittaro and Ranon, 2004] the abstractions are provided by hand by a human expert and just adapted by the algorithm, in our approach we automatically search for potential abstractions.

A solution to the problem of the automatic synthesis of component abstractions has been proposed in [Torta and Torasso, 2003]. The present paper extends the previous approach in a number of directions: first, we show how to the abstraction process is able to exploit contextual restrictions (the use contextual conditions have been advocated in [Dressler and Struss, 2003] for qualitative value abstraction); second, while in [Torta and Torasso, 2003] it is required that the model exhibits a directional and deterministic behavior, in the present paper we deal with non deterministic relational models; finally, in the present paper we are able to provide formal guarantees on the computational complexity of computing the abstraction mapping.

Also the authors of another recent proposal ([Siddiqi and Huang, 2007]) combine system abstraction with model compilation, using DNNF as the target formalism. The main difference w.r.t. the approach described in this paper is that the result of their algorithm is not a single model that replaces the original one, but a hierarchy of abstractions to be used at run-time to focus diagnosis. Moreover, their work focuses on combinatorial circuits (which are inherently directional) and on preserving just minimum cardinality diagnoses.

The experiments performed on the hydraulic system show that the approach is effective both in inferring interesting component abstractions and in terms of computational cost. So far, the criteria for rejecting an abstraction (is-good predicate) are relatively simple since they are based on the number of abstract behavioral modes. However, it is possible to extend the criteria used by is-good for including other parameters that could influence the suitability of an abstract component along the line of reasoning discussed in [Provan, 2001].

References


Author Index

A
Abreu, Rui ........................................... 7, 127
Alonso-González, Carlos J .................. 215, 339
Anselma, Luca ................................... 189
Armant, Vincent .................................. 223
Armengol, Joaquim .............................. 261

B
Bayoudh, Mehdi ................................ 231
Benazera, Emmanuel .......................... 15, 23
Biswas, Gautam ................................. 39, 47, 151, 261
Blackhall, Lachlan .............................. 239
Brandán Briones, Laura ...................... 31
Bregón, Aníbal ................................. 39, 339

C
Castillo, Sandra .................................... 261
Chanthery, Élodie ............................... 15
Combacau, Michel .............................. 269, 347

D
Dague, Philippe .................................. 31, 223
Daigle, Matthew ................................. 47
de Kleer, Johan .................................. 55, 63, 109, 247, 291
Do, Minh ........................................... 63, 109

E
Esfandiari, Babak ................................. 253

F
Feldman, Alexander ............................ 71
Fesq, Lorraine ................................. 3
Frisk, Erik ........................................... 79

G
Gagnon, François .............................. 253
Gelso, Esteban R. ............................... 261
Goupil, Philippe ................................. 323
Grastien, Alban .................................. 95

H
Heintz, Fredrik .................................... 79

I
Ingham, Michel ................................... 3
Jauberthie, Carine .............................. 269, 323

K
Kalech, Meir ..................................... 87
Kaminka, Gal A .................................. 87
Kan John, Priscilla ............................. 95, 239
Karim, Jawad .................................... 269
Kather, Axel ...................................... 173
Kleiner, Alexander .............................. 275
Koutsoukos, Xenofon .......................... 47, 151
Kreum, Willibald ................................. 283
Krysander, Mattias .............................. 79, 101
Kuhn, Lukas ..................................... 63, 109, 291
Kurien, James .................................. 117

L
Lazovik, Alexander ............................. 31
Lindner, Michael ............................... 87
Liu, Juan ......................................... 291

M
Mayer, Wolfgang ............................... 127, 165
Mengshoel, Ole .................................. 307
Micalizio, Roberto .............................. 299
Myllymäki, Petri ................................. 143

N
Narasimhan, Sriram ............................. 307
Nica, Mihai ....................................... 135
Nyberg, Mattias ................................. 101, 143, 181, 197

O
Olive, Xavier .................................... 231

P
Peischl, Bernhard ............................... 315
Pencolé, Yannick ....................... 331, 347
Pernestål, Anna ................................. 143
Pons, Renaud .................................. 323
Price, Bob ....................................... 63, 109, 291
Prieto, Oscar J. .................................. 215
Provan, Gregory ............................... 71
Pucel, Xavier .................................... 331
Pulido, Belarmino .............................. 39, 215, 339
R
R-Moreno, María Dolores ........................................ 117
Riaz, Naveed .................................................. 315
Ribot, Pauline ................................................ 347
Rodríguez, Juan J. .......................................... 215
Roll, Jacob .................................................... 79
Roychoudhury, Indranil ................................. 151

S
Sachenbacher, Martin ..................................... 159
Schneider, Dominik ..................................... 173
Schumann, Anika ........................................ 165
Schwoon, Stefan .......................................... 159
Silander, Tomi ............................................. 143
Simon, Laurent ........................................... 223
Steinbauer, Gerald ....................................... 275
Struss, Peter ................................................. 173
Stumptner, Markus ........................................ 127, 165
Svärd, Carl .................................................. 181

T
Theseider Dupré, Daniele ................................. 189
Torasso, Pietro ............................................. 299, 355
Torta, Gianluca ............................................ 189, 355
Travé-Massuyès, Louise .......................... 23, 231, 323, 331

V
van Gemund, Arjan J.C. .............................. 7, 71, 127
Voigt, Tobias ............................................... 173

W
Warnquist, Häkan ......................................... 197
Weber, Jörg ................................................. 135, 205
Wettig, Hannes .......................................... 143
Wotawa, Franz ............................................. 135, 205, 275, 283, 315

Z
Zhou, Rong .................................................. 63, 109, 291
Zoeteweij, Peter .......................................... 7

Alban Grastien, Wolfgang Mayer, and Markus Stumptner, Editors.
Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08),
September 22–24, 2008, Blue Mountains, NSW, Australia.