Proceedings of

The 24th International Workshop on Principles of Diagnosis

Edited by
Alexander Feldman
Meir Kalech
Gregory Provan

Contents

Organizing Committee

Meir Kalech	Ben-Gurion University of the Negev
Alexander Feldman	Nspyre, General Diagnositcs, and Delft University of Technology
Gregory Provan	University College Cork

Program Committee

Rui Abreu	University of Porto
Anibal Bregon	University of Valladolid
Steve Chien	NASA JPL
Luca Console	University of Turin
Matthew Daigle	NASA Ames Research Center
Johan de Kleer	Palo Alto Research Center
Teresa Escobet	Universitat Politécnica de Catalunya
Erik Frisk	Linkpings Universitet
Alban Grastien	NICTA and the Australian National University
Mitch Ingham	NASA JPL
Eli Kh	Ben-Gurion University of the Negev
Jan Eric Larsson	GoalArt & Lund University
Peter Lucas	Radboud University
Elizabeth Marie Massey	United Technologies Research Center
Louise Trave Massuyes	French National Center for Scientific Research
Wolfgang Mayer	University of South Australia
Pieter Mostermann	MathWorks
Sriram Narasimhan	UC Santa Cruz @ NASA Ames Research Center
Mattias Nyberg	Scania and KTH
Barry OSullivan	University College Cork
Bernhard Peischl	Softnet Austria
Yannick Pencolé	CNRS-LAAS
Belarmino Pulido	University of Valladolid
Lior Rokach	Ben-Gurion University of the Negev
Indranil Roychoudhury	SGT Inc. @ NASA Ames Research Center
Martin Sachenbacher	Technische Universität München
Anika Schumann	IBM
Yuval Shahar	Ben-Gurion University of the Negev
Roni Stern	Harvard
Markus Stumptner	University of South Australia
Sylvie Thiebaux	ANU & NICTA
Gianluca Torta	University of Turin
Arjan van Gemund	Delft University of Technology
Cees Witteveen	Delft University of Technology
Franz Wotawa	Technische Universität Graz
Marina Zanella	University of Brescia

Foreword

Welcome to the twenty-fifth meeting of the International Workshop on Principles of Diagnosis (DX), DX-2013. Unlike counting the years of the modern era, DX counting starts correctly at zero. DX is a workshop that provides a forum to promote detailed technical exchange and debate while at the same time it makes efforts to develop synergistic approaches to solving real-world problems. It is an annual event that started in 1989, originally rooted in the Artificial Intelligence (AI) community. Papers presented at the workshop cover a variety of theories, principles, and computational techniques for diagnosis, monitoring, testing, reconfiguration, fault-adaptive control, and repair of complex systems. Applications of these theories, principles, and techniques to industry-related disciplines and other real-world problems are also important topics of the workshop.

Like the previous workshops in this series, DX-2013 encourages the interactions and the exchange of theories, techniques, applications, and experiences among researchers and practitioners from different backgrounds: Artificial Intelligence, Control Theory, Systems Engineering, Software Engineering and other related areas, who share an interest in different aspects of diagnosis, and the related fields of testing, reconfiguration, maintenance, prognosis, and fault-adaptive control. Similar to previous meetings, we have adopted a single-track program with a limited number of participants in order foster close interchange of ideas.

Over the years, DX has expanded from its roots in AI to incorporate ideas from a wide range of other disciplines. Such disciplines include control (e.g., Fault Detection and Isolation and Discrete-Event Systems techniques), hybrid systems, modelling, etc. Because diagnosis/repair is a fundamental aspect of almost every real-world domain, it is important that all disciplines relevant to diagnosis/repair have the freedom to contribute to this workshop. The papers appearing at this Workshop reflect many aspects of the broad field of diagnosis/repair: we have tracks that cover "traditional" AI-based topics such as Conflicts, Diagnoses, and Hitting Sets, Control-oriented methods (dynamical and hybrid systems), as well as a range of Applications-focused techniques. In addition, the Diagnosis Competition, DXC, reflects the breadth of focus to which the Workshop has expanded: it has tracks for synthetic models (logic circuits), software, as well as real-world continuous-valued systems (power supplies and climate-control systems).

Fault diagnosis aims at identifying the root cause(s) of system malfunction based on external observations. For the purpose of diagnosis the system is broken down in a number of components (subsystems) in terms of which the diagnosis is expressed. The diagnostic process takes observations (values, events) as input, and produces a list of possible diagnoses, where each single diagnosis may involve multiple component faults to be a likely explanation for the observations (e.g., observations can be explained by the combination of components 13 and 99 at fault).

Due to the fact that the observations are typically limited in time and space (e.g., lack of measurement time, lack of sensors), and the fact that information on system behavior is limited, the size of the list of possible diagnoses can be large, while only one diagnosis represents the true system fault state. Next to computational complexity, this limited diagnostic accuracy is a key performance metric. Fault diagnosis is typically triggered by the detection of an error (or errors), i.e., the deviation of a system value from its nominal value, also referred to as a symptom.

Traditionally, both in diagnosis of software faults (defects, bugs) and hardware faults a symptom-based approach has been adopted, which is based on the existence of a mapping from symptoms to diagnoses. The implementation of this mapping can range from a symptom table to an expert system, and is typically compiled by humans. While such a mapping offers split-second diagnosis, the mapping

is usually not complete, introducing the risk of bad diagnostic performance in unforeseen situations. Moreover, deriving the mapping is an intensive and error-prone process given its complexity. Especially in software, implementing symptom-based diagnosis is virtually impossible, as illustrated by the fact that even simple exception handling (where diagnosing the root-cause of the exception is typically even left out) requires significant coding effort. The large cost associated with traditional diagnosis has lead to a number of contemporary approaches, which aim for completeness and automation.

The most comprehensive approach towards automatic diagnosis is Model-Based Diagnosis (MBD). In MBD a model of the system is used as reference to infer how each component is behaving (nominal, fault mode 1, fault mode 2, etc.). Given an observation, one essentially solves a system of constraints (model and observations) yielding a set of system health assignments (solutions) that are consistent with the model and observations. A popular approach within MBD is based on the use of propositional logic, which allows for efficient solvers. MBD is an active area of research, especially within AI research, and is almost entirely aimed at hardware fault diagnosis. Successful applications include NASA's Deep Space I probe and system diagnosis applications within ASML, and Océ, using the Lydia toolkit.

While MBD is well-suited for circuits and hardware devices, software is rarely modeled in sufficient detail during development, and derivation of suitable behavioral models from source code is troublesome at best. In fact, a model of a computer program is yet, although simpler, computer program. Hence, suitable to contain defects as well (Gödel paradox) Due the severe impact of software defects, much research has been performed on (semi-)automatic debugging. A recent and relatively successful approach to software fault diagnosis is Spectrum-based Fault Localization (SFL), which pairs very low complexity with good diagnostic performance, effectively outperforming work in the same complexity class. An example of such approach is Barinel that abstracts component behavior and reason with information on whether components have been executed in transactions or not. Due to the low diagnostic and modeling costs, SFL has enabled it to be applied to large, real software systems. A framework, GZoltar, providing such technique is available online.

Despite the recent advances in MBD and SFL, there is still work that needs to be done towards a framework that is able to deal with both software and hardware. A question that still remains to be answered is how to combine SFL with MBD techniques such as GDE and Lydia-NG. For example, take an hierarchical approach by first using SFL (because of its low complexity) and zoom-in at specific components - blamed by the SFL approach - with more expensive techniques such as GDE or Lydia-NG seems to be an interesting venue.

Alexander Feldman, Gregory Provan, Rui Abreu, and Meir Kalech

Regular Papers

Optimizing Model-Based Diagnosis Complexity for Analogue Linear Systems

Alexander Feldman and **Gregory Provan**
University College Cork, Cork, Ireland
e-mail: a.feldman@ucc.ie, g.provan@cs.ucc.ie

Abstract

Fault diagnosis of analogue systems is a challenging task and no fully automated solution exists. We address the task of diagnosing hard faults in linear analogue systems. We develop an algorithm that dynamically reduces the size of the simulation model during diagnostics inference. We compare our simulation approach to the approach used by industry and academia and empirically demonstrate that our algorithm provides significant speedups on a benchmark of analogue circuits with regular and semi-random topologies. We apply our algorithm to the model of a real-world satellite and demonstrate 40 times speedup. We measure a significant performance increase (200 times) for a configuration of the diagnostic search that achieves higher diagnostic accuracy, optimizing a trade-off in computational complexity versus diagnostic accuracy. Our framework can be directly applied to any system for which the bond graph energy formulation applies and the model optimization part of the algorithm can be used to improve the computational complexity and numerical stability of a large class of numerical solvers that require steady-state simulation as a part of their convergence process.

1 Introduction

Fault diagnosis of analogue systems is an important problem, given their prevalence in synthetic and natural domains. The most heavily-studied domain is that of analogue circuits, and enormous research has been directed towards the testing and diagnosis of electronic devices [Bandler and Salama, 1985].

We can separate three sub-tasks of analogue fault diagnosis: detecting faulty behaviors, isolating the faulty components, and determining the parameters of the faulty components. The success of this process depends on accurate definitions of fault models. We can classify fault models for analogue systems into two categories: hard (catastrophic) faults and soft (parametric) faults. A *hard (catastrophic) fault model* characterizes when the system exhibits significant (and often abrupt) behavioral changes. For example, a hard fault in a circuit occurs when the terminals of a component become stuck-open or stuck-short. A *parametric fault model* characterizes when there exist deviations of component parameters that result in performance beyond acceptable limits.

No fully-automatic method exists for multiple-fault diagnosis of analogue systems, even for the heavily-studied domain of circuit diagnosis. This paper proposes a method for diagnosis of hard multiple-fault instances in analogue systems. We adopt a model for analogue systems based on a graphical topological structure, in which nodes represent measurement points and edges the components. We propose an approach for diagnosing hard faults that performs topological reductions of the model during runtime inference, which results in significant speedups compared to existing approaches that do not employ such reductions [Korzybski, 2008].

The contributions of this paper are as follows: (1) we introduce a simulation-based search framework for diagnosis of analogue systems; (2) we design an optimization method that significantly improves the simulation performance in the case of failures; (3) we apply this approach to the domain of analogue electrical circuits; (4) we design a benchmark of analogue electrical circuits that represents a large class of circuit topologies; (5) we propose an analytical model that predicts speedups based on system topology; and (6) we experimentally show that the proposed optimization method improves the diagnostic performance as predicted by the analytical model.

2 Related Work

Analog systems diagnosis is a well-studied area, and a variety of approaches have been developed, based on methods such as off-line numerical test-generation [Duhamel and Rault, 1979], bond graph approaches [Samantaray *et al.*, 2006], machine learning [Aminian and Aminian, 2000], and AI-based methods [Dague *et al.*, 1992]. The domain of analogue systems that has received the most attention is that of fault diagnosis of analog circuits, e.g., [Bandler and Salama, 1985]. Although much progress has been made, most prior work addresses only single-fault cases. Achievements towards automatic multiple-fault diagnosis are documented in, i.a., [Korzybski, 2008]; however, many aspects of the problem are still open, and no fully-automatic method exists for multiple-fault generic analog circuit diagnosis.

Researchers have studied dynamic discontinuities in bond graphs [Mosterman and Biswas, 1996]. This paper complements the approach of Mosterman and Biswas by providing an algorithmic framework for dealing with structural (or parametric) discontinuities. Our approach is fully applicable to bond graphs [Mosterman and Biswas, 1999].

This work differs from the circuit topology-modification algorithms based on qualitative circuit models (e.g., [Hotz *et al.*, 1997]). In this qualitative circuit work, a qualitative structural model of a circuit is transformed (using star-delta transformations and series-parallel reductions) to generate a structure more suitable for fault simulation. The original approach, the SDSP method [Hotz *et al.*, 1997], was limited to resistive networks that consist of one voltage source and an unlimited number of resistors, but was generalized in [Snooke and Lee, 2013]. The circuit topology transformations in the SDSP method are performed *prior to* any inference (as opposed to during the course of diagnostics inference in our approach), and do not cover the cases addressed in our approach, in which extremal values (0 or ∞) may occur in voltage or current. Further, the class of transformations, because they are for a different purpose, are quite different than the transformations employed in this article.

3 Concepts and Definitions

In what follows we present the basic concepts of circuit diagnosis.

3.1 System Description

A system consists of an inter-connected set of components. For example, in a circuit a component can be a resistor or capacitor, and connections are wires. We represent our model in terms of two parts, the connection topology, and the component equations. We represent the topology of a system using a graph G.

Definition 1 (Topology Graph)**.** Given a model M with components $COMPS = \{c_1, c_2, \cdots, c_n\}$, and component connections (junctions) $Z = \{z_1, z_2, \cdots, z_l\}$, where component c_i occurs between junctions $z_j,\ z_k$, we represent the topology graph $G(V, E)$ of M such that the vertices V correspond to connections in M and edges E correspond to components in M.

Edges in G are also labeled with the type of the corresponding components and their parameters. We represent the equations as follows. We represent a generic linear analogue system in terms of a relation between effort $\vec{x}$ and flow $\vec{z}$ vectors of variables, using $T\vec{x} = \vec{z}$, where T is an $n \times m$ matrix. For example, for circuits $\vec{x} \in \mathbb{R}^n$ is an (unknown) nodal voltage vector, and $\vec{z} \in \mathbb{R}^m$ is a measurable current-source vector.

We adopt a simulation approach called Modified Nodal Analysis (MNA) [Ho *et al.*, 1975]. This approach converts the component equations, together with the system topology, into a matrix representation that incorporates both equations and topology using the Kirchoff Current Law (KCL), in order to enable efficient simulation. Due to space constraints, we refer the reader to [Ho *et al.*, 1975] for a full description of MNA; here we focus on optimizations of MNA when using this approach for hard-fault diagnostics inference.

As an example of problem formulation using MNA, consider a simple circuit, as shown in Figure 1. In the circuit we identify 3 nodes at which we measure voltage, denoted $v_a,\ v_b,\ v_c$. This circuit has 2 voltage sources, $V_1,\ V_2$, and two current flows $i_{v1},\ i_{v2}$. The 3 nodes and 2 voltage sources, ($n = 3,\ m = 2$), result in a system with system equation $T\vec{x} = \vec{z}$ and characteristic nodal matrix T of size $(n+m) \times (n+m)$. The resulting matrix is shown in equation 1. For example, the first of 5 possible equations describing the system is obtained by applying KCL at v_a, i.e., $\frac{1}{R_1} v_a - i_{v1} = 0$. This is captured by the first row of the T matrix, and the first element of the x and z vectors.

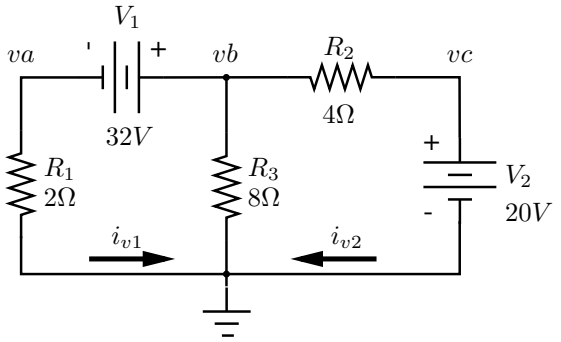


Figure 1: Simple circuit example

$$\begin{bmatrix} \frac{1}{R_1} & 0 & 0 & -1 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} & 1 & 0 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \\ i_{v1} \\ i_{v2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V_1 \\ V_2 \end{bmatrix} \quad (1)$$

From our general system equation $T\vec{x} = \vec{z}$, our example maps to equation 1 as follows. The T matrix denotes only known quantities; in our circuit example it denotes the values of the passive elements (the resistors). The vector $\vec{x}$ denotes the unknown quantities (efforts and flows); for our circuit these correspond to node voltages and the currents through the independent voltage sources. The vector $\vec{z}$ denotes only known quantities (efforts and flows).

3.2 Model-Based Diagnosis

We will adopt a mode-based representation, i.e., we assume that the system can operate in a set Ω of modes, which can consist of nominal or faulty operating conditions. Given a system that consists of a discrete set of components with a corresponding set of health parameters COMPS, a mode $\omega \in \Omega$ is an assignment to all variables in COMPS.

Further, for each mode we assume that we can specify a distinct set of equations. Hence, for each $\omega \in \Omega$ we specify an equation set SD_ω given by $T_\omega \vec{x_\omega} = \vec{z}_\omega$.

Definition 2 (Diagnostic Model)**.** Given a system that consists of a discrete set of components with a corresponding set of health parameters COMPS, a diagnostic model $M = \langle \mathrm{SD}, \mathrm{COMPS} \rangle$ is specified using a function $\mathrm{SD} = \bigcup_{\omega \in \Omega} \mathrm{SD}_\omega$.

In this article, we are typically given the flow vector $\vec{z}$ and must compute the effort vector via $\vec{x_\omega} = T_\omega^{-1} \vec{z}_\omega$, a process we call *simulation* of $\vec{x_\omega}$. Since SD is linear, we can simulate efficiently.

Given an observation $\vec{\alpha}$, we estimate the mode (i.e., solve a diagnostic problem) by computing an optimal solution of a parameter estimation problem where the parameters are discrete and the problem is split in two parts: simulation and residual analysis.

In real-world applications, straightforward simulation function (from definition 2) is not sufficient to adequately

solve the diagnostic problem. This is because models are imprecise, there is sensor noise, health parameters are discrete, etc. Instead, we compute a difference between $\vec{\alpha}$ and a simulation $\hat{\vec{z}}$ (using the residual function of Definition 3), and then identify the mode that minimises this function.

Definition 3 (Residual Function)**.** Given two m-dimensional real vectors $\hat{\vec{z}}, \vec{\alpha} \in \mathbb{R}^m$, a residual function $R : \{\hat{\vec{z}}, \vec{\alpha}\} \mapsto R(\hat{\vec{z}}, \vec{\alpha})$ maps $\hat{\vec{z}}$ and $\vec{\alpha}$ into the real interval $[0; \infty)$.

For the residual function R we typically use some statistical estimator. In this paper we use the absolute residuals function:

$$R_{\text{abs}}(\hat{\vec{z}}, \vec{\alpha}) = \sum_{i=1}^{m} w_i \left| \hat{z}_i - \alpha_i \right| \tag{2}$$

where $\hat{z}_i$ and α_i are the i^{th} components of the $\hat{\vec{z}}$ and $\vec{\alpha}$ vectors, respectively, and w_i are weights (parameters).

Definition 4 (Health Estimation Problem)**.** Given a diagnostic model SD and a residual function R, the health estimation problem is to compute an assignment $\omega_{\min}$ to all variables in COMPS such that:

$$\omega_{\min} = \operatorname*{argmin}_{\omega} R(\text{SD}_\omega, \vec{\alpha})$$

Solving the above health estimation problem, while maintaining computational efficiency is the main goal of our framework.

4 Algorithms

In what follows we outline an algorithm for solving the problem given in definition 4. Our algorithm consists of (1) a search algorithm that iterates over a subspace of the possible combinations of discrete parameter values, (2) a model optimization algorithm that improves the model complexity, and (3) simulation and residual algorithms that rank the candidates and compute the optimal parameter assignment. The primary goal of this section is to show how model optimization (2) significantly improves the performance of the simulation step (3).

Algorithm 1 provides a generic search method for solving the parameter estimation problem in definition 4. The idea is to search the space of all possible health parameter combinations. This space is exponential in the number of components (health variables) and normally the search is limited to a subset of all combinations. Examples of such subsets are all k-fault combinations or the assignment sets generated by a greedy search [Feldman *et al.*, 2010].

Algorithm 1 starts by assuming the user-defined default parameter assignment $\mathbf{h} = \{\}$ which, in diagnostic context, represents the *all-okay* status of the system. Successive health assignments are generated by the NEXTHEALTHASSIGNMENT subroutine.

Depending on the implementation of NEXTHEALTHASSIGNMENT, algorithm 1 can perform different types of search such as breadth-first, depth-first, iterative deepening, greedy (in this case NEXTHEALTHASSIGNMENT uses the r parameter), random, and many others. Which policy is optimal depends, i.a., on the simulation and residual functions (see def. 3) and the topology of the model. Choosing a search policy (or a combination of such) is a topic of its own and is not discussed in this paper. Our main focus is how the complexity of the simulation function SIMULATESYSTEM

Algorithm 1: DIAGNOSISSEARCH(M, α)

Input: M, model
Input: α, observation
Result: ω, diagnosis
Local variable: $\mathbf{h}$, health assignment, initially $\{\}$
Local variable: $\vec{p}$, simulation vector
Local variables: $r, r_{\min}$, residuals, $r, r_{\min} \in [0; \infty)$

```
1 r ← ∞
2 repeat
3 |   p⃗ ← SIMULATESYSTEM(MAKENETLIST(M, h))
4 |   r ← COMPUTERESIDUAL(p⃗, α)
5 |   if r < r_min then
6 |   |   ω ← h
7 |   |   r ← r_min
8 until h ← NEXTHEALTHASSIGNMENT(h, r) ≠ ∅;
9 return ω
```

in line 3 affects the overall complexity of the optimization algorithm.

The function MAKENETLIST takes a set of component models, a topology, and a parameter guess $\mathbf{h}$ and generates a netlist. This netlist is fed to the simulation function SIMULATESYSTEM. The netlist represents a system of linear equations and the simulation subroutine, described in the section that follows, uses linear algebra tools to compute the unknown simulation variables.

Algorithm 2 outlines a circuit simulator that supports only linear elements: dissipative elements (resistors), effort (voltage) and flow (current) sources. Our approach is similar to the one used in SPICE [Nagel and Pederson, 1973]. The implementation of Algorithm 2 uses the Modified Nodal Analysis (MNA) of Ho *et al.* [1975], generating nodal matrices of size $n \times n$ where n is the number of nets in the input netlist L.

Algorithm 2: SIMULATESYSTEM(L)

Input: L, netlist
Result: $\vec{z}$, voltage vector
Local variable: G, graph
Local variable: N, nodal matrix
Local variable: $\vec{j}$, current vector

```
1 G ← MAKEGRAPH(L)
2 G ← OPTIMIZEGRAPH(G)
3 N ← MAKENODALMATRIX(G)
4 j⃗ ← MAKECURRENTVECTOR(G)
5 z⃗ ← N^{-1} j⃗
6 return z⃗
```

Algorithm 2 starts by converting the input netlist L into a graph G. This is achieved by calling the MAKEGRAPH subroutine. The implementation of MAKEGRAPH is not discussed in this text because a netlist, describing a linear circuit, is already almost a graph, and constructing G from L is straightforward.

As is customary in SPICE, elements in L are represented as edges in G and nets in L are nodes in G. Edges in G are labeled with the type of the corresponding elements and their parameters.

The matrix N is directly generated from the graph G as described by Kielkowski [1994, p. 18]. The MNA approach, however, does not tolerate short-circuits as they lead

to infinite conductances. Further, open-circuits may cause singular matrices. To avoid these problems, even modern solvers replace short-circuits with small and open-circuits with large resistances. In this paper, resistances close to zero and large resistances are denoted as ε and $\mathcal{E}$, respectively. While replacing zeroes and infinity with small and large numbers is less of a problem for a single simulation, it increases significantly the overall complexity when solving thousands of times for various combinations of parameter values.

The time complexity of algorithm 2 is dominated by the matrix inversion in line 4. The matrix inversion operation is as complex as matrix multiplication [Cormen *et al.*, 2001, p. 757] and its complexity is $O(n^3)$ or $O(n^{2.807})$ when using the Strassen algorithm[1] [Press *et al.*, 2002, pp. 105–107] where the matrix N is of size $n \times n$.

The most important addition to algorithm 2 is the OPTIMIZEGRAPH call in line 2. The purpose of OPTIMIZEGRAPH is to reduce the number of nodes in G, thus reducing the size of the nodal matrix N.

The task of algorithm 3 is to reduce the complexity of simulation by removing components with specific parameters from the model. Algorithm 3 has two main parts: first all components that are faulty (for example open- and short-circuited resistors) are removed (lines 3–10) and, second, only those circuits that are closed through a voltage source are retained (lines 12–24).

Algorithm 3: OPTIMIZEGRAPH(G)

Input: $G = \langle V, E \rangle$, graph
Result: G, optimized graph
Local variables: v, vertex, e, edge
Local variables: S, edge stack, P, set of edges
Local variable: *stop*, Boolean flag

```
1  repeat
2      stop ← true
3      foreach e ∈ E : ISDISSIPATIVEELEMENT(e) do
4          if Ω(e) < ε ∨ SRC(e) = DST(e) then
5              V ← V ∖ DST(e)
6              E ← E ∖ e
7              RECONNECT(E, SRC(e), DST(e))
8              stop ← false
9          if Ω(e) = ∞ then
10             E ← E ∖ e
11 until stop = false;
12 PUSH(s, EFFORTSOURCE(E))
13 while e ← POP(s) ≠ ∅ do
14     PUSH(s, ADJACENTEDGES(G, e))
15     P ← P ∪ e
16 repeat
17     stop ← true
18     foreach e ∈ E do
19         if |e| < 2 or e ∉ P then
20             V ← V ∖ DST(e)
21             E ← E ∖ e
22             stop ← false
23 until stop = false;
24 REMOVEORPHANEDVERTEXES(V, E)
25 return G = ⟨V, E⟩
```

[1] There exist faster methods for sparse matrices but they are of no practical significance for our algorithms.

The auxiliary function ISDISSIPATIVEELEMENT(e) returns true if and only if its argument e is a dissipative element edge (resistor in the electrical domain). The function $\Omega(e)$ returns the resistance of e. The functions SRC(e) and DST(e) return the two respective vertexes that are connected to an edge e. Notice that it is customary that the representation of the simulation graph V is directed, although the orientation of the edges produces no difference in the simulation results.

While edges that represent open-circuited resistors can be simply removed from the graph (lines 9–10), after removing a short-circuited resistors (lines 4–8), the adjacent wires have to be reconnected. This is done by the RECONNECT subroutine and the process is illustrated in figure 2. As removing a short-circuit can cause another short-circuit (see again fig. 2), the component removal loop in lines 3–10 has to be repeated until no more removals are performed. This is achieved by using the *stop* flag.

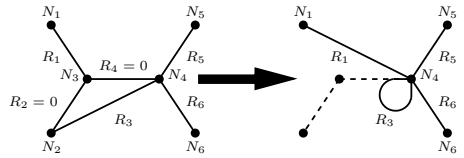


Figure 2: Removing the short-circuited resistors R_2 and R_4 in the left graph makes nodes N_2 and N_3 unnecessary. To preserve the circuit we have to disconnect R_1 from N_2 and to connect it to N_4. Similarly, both ends of R_3 have to be connected to N_4. At this moment, N_4 shorts R_3 and R_3 can be removed.

The second part of algorithm 3 (lines 12–22), removes all circuit elements that are not doubly-connected to a voltage source. This process first performs a Depth-First Search (DFS) on the graph G [Sedgewick, 2002, pp. 81–99]. In our case, the DFS starts from all voltage sources. The subroutine EFFORTSOURCE returns all edges that are connected to a positive terminal of an effort (voltage) source. These edges are added to the stack s in line 12. The result of the DFS is that all edges that can be reached from a voltage sources are added to the set of edges P. The DFS uses the function ADJACENTEDGES to generate all edges that are neighbors of an edge e.

The loop in lines 16–23 removes all edges that are not in P, i.e., they are not connected to a voltage source. This loop also removes all hanging edges. The condition for a hanging edge is in line 19. The expression $|e|$ denotes the sum of the degrees of the two vertexes incident to e. Similarly to the first part of the algorithm, the loop in lines 18–22 is repeated until no further truncation of the graph is possible.

Finally, all orphaned nodes (zero-degree nodes) are removed by the helper subroutine REMOVEORPHANEDVERTEXES.

The worst-case time complexity of algorithm 3 is $O(|E|^2)$, i.e., it is polynomial in the number of edges. Some time complexity in algorithm 3 can be traded for memory, thus achieving near linear performance. The outer loops in lines 1–11 and 16–23, for example, can be removed at the cost of managing data structures that keep track of all graph nodes that have to be merged and all orphaned or hanging paths.

5 Average-Case Analysis

This section examines the type of complexity reduction that is possible for a typical model.

Definition 5 (Topology Graph)**.** Given a model M with components COMPS = $\{c_1, c_2, \ldots, c_n\}$, and junctions Z = $\{z_1, z_2, \ldots, z_l\}$, where component c_i occurs between junctions $z_j,\ z_k$, we map M into a graph $G(V, E)$ of vertices V and edges E by mapping each component to an edge and each junction to a vertex.

Parallel edges are allowed. The size of a graph $G(V, E)$ is denoted[2] as $|G(V, E)|$ where $|G(V, E)| = |E|$. The size of $G(V, E)$ is essential to the algorithmic performance of algorithm 3 as it is equal to the size of the nodal matrix.

Definition 6 (Contraction Operators)**.** Given a graph $G(V, E)$ and an edge $e = \{v, w\}$, the graph $G'(V', E') = G(V, E) \searrow e$ is such that $E' = E \backslash \{e \cup F\}$, $V' = V \backslash v$, F is the set of all edges that are parallel to $\{v, w\}$, and each edge $\{x, v\} \in E$ such that $x \neq w$ is replaced with an edge $\{x, w\} \in E'$.

Given a graph $G(V, E)$ and an edge e, the graph $G'(V', E') = G(V, E) \nwarrow e$ is such that $E' = E \backslash e$, $V' = V \backslash \{W \cup v\}$ where v is incident to e and W is the set of singly connected vertices in $E \backslash v$.

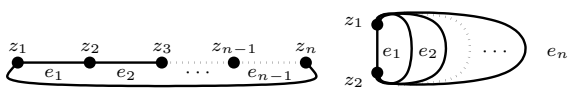
We have chosen the symbols $\searrow$ and $\nwarrow$ to visually resemble the set exclusion operator $\backslash$ as the two contraction operators lead to a decrease in the size of G. The graph contraction ratio is:

$$\rho = \frac{\sum_{e \in E} \left[|G(V, E) \searrow e| + |G(V, E) \nwarrow e| \right]}{2 |G(V, E)|^2} \tag{3}$$

The variable ρ in eq. 3 adds-up the effect of applying $\searrow$ and $\nwarrow$ to each edge in E. Note that $\rho \in [0; 1]$. The denominator in eq. 3 is simply $|G(V, E)|$-times the number of edges in $G(V, E)$. The coefficient 2 in the denominator is because we have two contraction operators: $\searrow$ and $\nwarrow$. In the empty graph, $\rho = 1$ by definition, i.e., no contraction is possible. In graphs with a single loop, $\rho = 0$, i.e., the sum of the sizes of all contracted graphs is zero.

Lemma 1. *The following two statements are true:*

1. *The total graph contraction for a serial graph $G(V, E)$ with edges $E = \{\{z_1, z_2\}, \{z_2, z_3\}, \ldots, \{z_{n-1}, z_n\}\}$ is $\rho = \frac{1}{2}$.*
2. *The total graph contraction for a parallel graph $G(V, E)$ with edges $E = \{\{z_1, z_2\}, \{z_1, z_2\}, \ldots, \{z_1, z_2\}\}$ is $\rho = \frac{1}{2}$.*



(a) serial (b) parallel

Figure 3: Boundary graph topologies

[2]This is an abuse of notation. Bollobás [1998], for example, uses $|G|$ to measure the order of $G(V, E)$, while the graph size is denoted as $e(G)$.

Proof (Sketch). We show the correctness of statement 1 and statement 2 can be proven in a simmilary way.

In a graph $G(V, E)$ with serial topology, there are exactly $G(V, E) = |E| = n$ edges as illustrated in figure 3(a). Choose any edge e. Applying $\searrow$ to e, removes only e from the original graph $G(V, E)$ and results in:

$$\sum_{e \in E} |G(V, E) \searrow e| = n\,(n-1) \tag{4}$$

On the other hand, applying $\nwarrow$ to e, regardless of the choice of e results in a singly-connected graph:

$$\sum_{e \in E} |G(V, E) \searrow e| = n \tag{5}$$

Adding eq. 4 to eq. 5 gives us $\rho = \frac{n+n(n-1)}{2n^2}$ which simplifies to $\rho = \frac{1}{2}$. □

We next continue with analyzing the effect of algorithm 3 on random graphs. The subject of this analysis are random graphs, of the Erdős and Rényi type [1960], well below the phase transition, where the components are serially connected. Due to serially connecting all components, we refer to these graphs as "semi-random".

We denote the average graph degree of a graph $G(V, E)$ as $\bar{d}$ where $\bar{d} = 2|E|/|V|$.

Theorem 1. *The total graph contraction for a serially-connected semi-random graph $G(V, E)$ is $\rho = k\bar{d}$.*

Proof (Sketch). We decompose $G(V, E)$ into two graphs: $G(V, E')$ and $G(V, E'')$ where E' is the set of random edges and $G(V, E'')$ is a serially connected graph such as the one shown in figure 3(a). This decomposition is possible due to the way $G(V, E)$ is constructed. Let $G(V, E'')$ consists of C components (the computation of C is shown in the paper of Erdős and Rényi [1960]). We can now show that the overall reduction $\rho = k_1\rho' + k_2\rho''$ will be the average of reducing $G(V, E')$ and $G(V, E'')$. From lemma 1 we have that $\rho'' = \frac{1}{2}$. Finally, both k_1 and k_2 are proportional to $\bar{d}$, hence $\rho = k_1'\bar{d}\rho' + k_2''\bar{d}\rho''$. As there is no dependency between k_1', k_1'', ρ', and ρ'' (ρ'' depends on C only), then we can conclude that ρ is proportional to the average graph degree. □

Although there is an analytical method to obtain ρ for semi-random graphs, we compute it experimentally which also shows the absolute improvement in computational complexity due to the symbolic preprocessing. This we do in the section that follows next.

6 Experiments

We have run a big number of experiments to characterize the performance of the algorithms described in section 4.

Algorithms 1–3 are implemented as a part of the (deleted for anonymity) diagnostic framework. The implementation is in C++ and uses the BOOST C++ library collection [Schäling, 2011].

6.1 Benchmark

Unlike with digital circuits [Brglez and Fujiwara, 1985] and to the best of our knowledge, there is no linear SPICE benchmark, so we have generated the circuits that we need

Table 1 shows a number of regular circuits. These topologies can be scaled by setting a variable N, producing a range of circuit sizes.

Name	N	variables	|COMPS|
N-SERIAL	3–202	11–608	3–202
N-PARALLEL	3–202	9–407	3–202
N-MIXED	3–102	18–513	6–204
N-MESH	3–12	48–723	18–288
N-TREE	3–5	57–6253	27–3125

Table 1: Circuit benchmark

All benchmark circuits have a single voltage source that cannot fail. The N-SERIAL circuits, shown in figure 4(a), consist of N resistors connected in series. Similarly, the N-PARALLEL circuits in figure 4(b) consist of N resistors connected in parallel. The N-MIXED topology is a combination of N-SERIAL and N-PARALLEL as shown in figure 4(c). The N-MESH circuits consist of $2 \times N^2$ resistors arranged in a rectangular grid as shown in figure 4(d). Finally, the N-TREE circuits are complete N-ary trees of depth N. They are shown in figure 4(e).

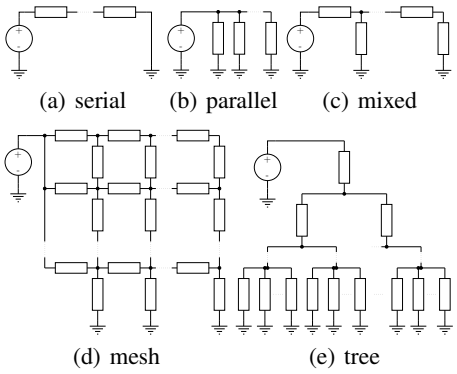


Figure 4: Scalable circuits with regular topology used for performance analysis

For N-SERIAL, $\bar{d} = 2$. For N-PARALLEL, $\bar{d} = N + 1$. For N-MIXED, $\bar{d} = (4N + 2)/(N + 2)$ which approaches 4 for larger N. The N-MESH circuits have average graph degree $\bar{d} = (4N^2 + 2)/(N^2 + 2)$ which approaches 4 when increasing N even faster than N-MIXED.

In addition to the regular circuits from the preceding section we have created 6 370 semi-random circuits. These circuits are generated by algorithm that is an adaptation of the random graph algorithm proposed by Sedgewick [2002, p. 42]. The reason for the modification is that the original algorithm produced multiple graph components for sparse graphs.

We generate random graphs by specifying the vertex/edge ratio r. The relation between the semi-random graph generation paramter r and the average graph density $\bar{d}$ is given by $|E| = rN + C - 1$ where C is the number of connected components in G. From the fact that $|V| = N$ it follows that $\bar{d} = 2Nr + C - 1/N$. For our experiments we have chosen $0.1 \le r \le 0.4$ which translates to $2.06 \le \bar{d} \le 2.9$.

6.2 Results

The performance gain for N-PARALLEL is the same as for N-SERIAL. Not surprisingly, algorithm 3 helps only a little in the connected topologies of figure 4(c) and figure 4(d). For these two dense topologies, we have measured a performance gain of at most 9%. This is because a single faulty component decreases the size of the nodal matrix by one.

The most significant benchmark performance gain due to algorithm 3 is for k-fault simulations where $k > 1$. The performance of the diagnostic search decreases exponentially with k, except when k approaches N because then the nodal matrix is almost degenerate, hence easier to decompose. On the other hand, multiple faults increase the frequency with which significant parts of the nodal graphs are pruned. For example, when $k = 2$, an open-circuit close to the voltage source in N-SERIAL makes N simulations use a 2×2 matrix instead of the normal $N \times N$.

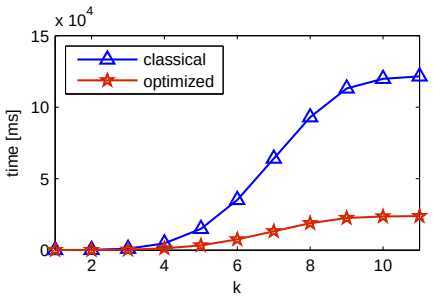


Figure 5: 11-SERIAL k-fault performance

The performance gain due to k-fault simulations is shown in figure 5. In order to simulate k-fault combinations, we have chosen a small $N = 11$. The performance gain increases for larger k and reaches a factor of 5.12 for $k = 12$.

Figure 6 shows the average algorithm complexity with random circuits. On the x-axis we have the number of nodes in the topology graph. The y-axis is the number of edges coefficient r. The z-axis shows the ratio of the time of the diagnosis with and without algorithm 3.

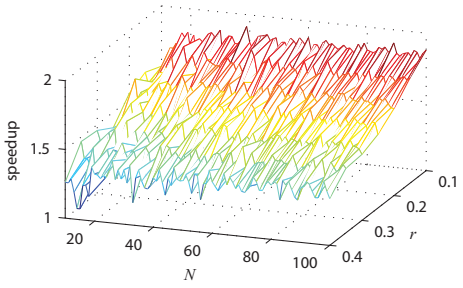


Figure 6: Performance gain for random circuits

Figure 6 shows that the performance gain due to algorithm 3 does not depend on the circuit size, i.e., it is almost constant. What determines the performance gain is the edge density. For $r = 0.1$ we have almost twice speed improvement and it decreases to ≈ 1.25 for $r = 0.4$. The performance gain is bounded from below by the performance gain of the N-SERIAL topology due to the nature of random graph generation algorithm.

The linear correlation coefficient of the speedup shown in figure 6 and the average graph degree is -0.89 which validates the analysis in section 5. The negative sign in the correlation is due to the fact that r defined as the number of graph vertices per edge while $\bar{d}$ is reciprocal to r, i.e., $\bar{d}$ is defined as the number of edges per graph vertex.

We have a modeled the electrical power system of a real-world satellite [deleted, 1900]. It consists of, a.o, 96 heating elements (modeled as resistors) and 112 switches (modeled as resistors whose resistor changes as a result of a user-supplied command). The large number of switching components and the designed cold-redundancy leads to a large number of normally open-circuit or short-circuit elements at any given time. As a result, algorithm 3 leads to a performance speedup factor of at-least 38.9, and that is for the single fault assumption. For the double-fault assumption we have a performance gain of 194 times.

7 Conclusions

In this paper we study the advantages of preprocessing the simulation model for steady-state analysis in diagnosis. The speedup due to pruning of parametrized components that cause discontinuity in the model depends on the topology of the model and we have established that throughout extensive experimentation on a benchmark of circuits. Our method does not decrease the diagnostic accuracy.

We observe most computational savings in real-world circuits where, due to standard redundancy for fault-tolerance, there are many unpowered and shorted sub-circuits that can be pruned. We also observe that the computational performance increases for higher k when considering k-combination of faults.

Acknowledgments

One of the authors has been supported in the writing of this paper by SFI grant 12/RC/229.

References

[Aminian and Aminian, 2000] Mehran Aminian and Farzan Aminian. Neural-network based analog-circuit fault diagnosis using wavelet transform as preprocessor. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(2):151–156, 2000.

[Bandler and Salama, 1985] John W. Bandler and Aly E. Salama. Fault diagnosis of analog circuits. *Proc. of the IEEE*, 73(8):1279–1325, 1985.

[Bollobás, 1998] Bélla Bollobás. *Modern Graph Theory*. Springer, 1998.

[Brglez and Fujiwara, 1985] Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. ISCAS'85*, pages 695–698, 1985.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[Dague *et al.*, 1992] Philippe Dague, Philippe Devés, Pierre Luciani, and Patrick Taillibert. Analog systems diagnosis. In *Readings in Model-Based Diagnosis*, pages 229–234. Morgan Kaufmann, 1992.

[deleted, 1900] deleted. Deleted for anonymity. 1900.

[Duhamel and Rault, 1979] Pierre Duhamel and Jean-Claude Rault. Automatic test generation techniques for analog circuits and systems: A review. *IEEE Transactions on Circuits and Systems*, 26(7):411–440, 1979.

[Erdős and Rényi, 1960] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. In *Publication of The Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[Feldman *et al.*, 2010] Alexander Feldman, Gregory M. Provan, and Arjan J. C. van Gemund. Approximate model-based diagnosis using greedy stochastic search. *J. Artif. Intell. Res. (JAIR)*, 38:371–413, 2010.

[Ho *et al.*, 1975] Chung-Wen Ho, Albert E. Ruehli, and Pierce A. Brennan. The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems*, 22(6):504–509, June 1975.

[Hotz *et al.*, 1997] Lothar Hotz, Heiko Milde, Ralf Möller, and Bernd Neumann. Resistive networks revisited: Exploitation of network structures and qualitative reasoning about deviations is the key. In *Proc. 8th Int. Workshop on Principles of Diagnosis,(DX97)*, 1997.

[Kielkowski, 1994] Ron M. Kielkowski. *Inside SPICE: Overcoming the obstacles of circuit simulation*. McGraw-Hill, 1994.

[Korzybski, 2008] Marek Korzybski. Dictionary method for multiple soft and catastrophic fault diagnosis based on evolutionary computation. In *Proc. ICSES'08*, pages 553–556. IEEE, 2008.

[Mosterman and Biswas, 1996] Pieter J. Mosterman and Gautam Biswas. A formal hybrid modeling scheme for handling discontinuities in physical system models. In *Proc. AAAI/IAAI'96*, pages 985–990, 1996.

[Mosterman and Biswas, 1999] Pieter J. Mosterman and Gautam Biswas. Diagnosis of continuous valued systems in transient operating regions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 29(6):554–565, 1999.

[Nagel and Pederson, 1973] Laurence W. Nagel and Donald O. Pederson. SPICE (simulation program with integrated circuit emphasis). Technical Report ERL-M382, EECS Department, University of California, Berkeley, April 1973.

[Press *et al.*, 2002] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, 2002.

[Samantaray *et al.*, 2006] Arun K. Samantaray, Kamal Medjaher, Belkacem Ould Bouamama, Marcel Staroswiecki, and Geneviéve Dauphin-Tanguy. Diagnostic bond graphs for online fault detection and isolation. *Simulation Modelling Practice and Theory*, 14(3):237–262, 2006.

[Schäling, 2011] Boris Schäling. *The Boost C++ Libraries*. XML Press, 2011.

[Sedgewick, 2002] Robert Sedgewick. *Algorithms in C - Part 5: Graph Algorithms*. Addison-Wesley, 2002.

[Snooke and Lee, 2013] Neal Snooke and Mark Lee. Qualitative order of magnitude energy-flow-based failure modes and effects analysis. *Journal of Artificial Intelligence Research*, 46:413–447, 2013.

Diagnosis of Hybrid Systems by Consistency Testing

Alban Grastien[1,2]
[1]Optimisation Research Group, NICTA, Australia
[2]Artificial Intelligence Group, Australian National University, Australia
e-mail: first.last@nicta.com.au

Abstract

We propose a new approach of diagnosis of hybrid systems using SMT (SAT Modulo Theory) technology. Diagnosis of hybrid systems is reduced to a series of diagnosis questions that are solved using SMT solvers. We show that this approach allows to deal with a new class of systems and paves the way to a new approach to diagnosis of hybrid systems.

1 Introduction

We consider the problem of diagnosis of hybrid systems, i.e., dynamic systems that involve both continuous (e.g., temperature) and discrete – but possibly on continuous domains – (e.g., voltage, switch position) evolutions.

1.1 Related Work

Hybrid systems have been extensively studied in the literature. They are sometimes designated by the expression "multiple mode systems," where the set of modes is the domain of the discrete variables and each mode defines a different set of rules for the continuous evolution.

There are essentially two classes of diagnostic approaches for hybrid systems. The first class builds on the redundancies in the system model: under certain assumptions on the system mode (for instance, no fault in a specific part of the system), the model equations allow to derive a constraint (or *indicator*) on the observed variables, such that the violation of this constraint is indicative of a violation of one of the assumptions [Staroswiecki and Comtet-Varga, 2001]. The main issue addressed by the research community for this type of approach is that of the number of indicators, in particular because the set of system modes is the Cartesian product of its constituent modes and is consequently of exponential size. The popular strategy is to instantiate the indicators on the fly [Heintz *et al.*, 2008]. Other issues include flexibility: because the indicators are built for a given observability, they cannot accomodate optimally a change of observability (for instance, different sampling rates, temporary masking, etc.) and the automatic generation of the indicators when the model involves non reversible functions.

The second class of algorithms is based on the simulation of the hybrid system either tentatively exhaustively (multiple Kalman filters [Blom and Bar-Shalom, 1988]) or through Monte Carlo sampling (particle filters [Arulampalam *et al.*, 2002]). Given a probabilistic belief state, the possible evolutions of the state are computed and compared to the actual observation to update their respective probabilities. Again a major problem of this type of approach is that the number of evolutions is too large to enumerate and a substantial body of work is dedicated to reducing the representation of the belief state through approximation or pruning [Blom and Bar-Shalom, 1988; Benazera and Travé-Massuyès, 2009]. Such operations are however very hazardous because the correct assumption of unlikely events such as faults must often go through a steady-increasing probability period where the probability is still so low that it may be reset by the approximation process. Furthermore, this approach necessitates a model from which the state evolution can be predicted. In case of unknown behavioural mode, the best that can be done is to ignore the variables whose value cannot be predicted [Hofbaur and Williams, 2004].

Diagnosis of hybrid systems with SMT solvers was first suggested by Ernit and Dearden [2011]. The main differences with the work presented here is that i) they limit themselves to real-valued variables and do not discuss the dynamics of continuous variables, and ii) they only consider a conflict-based approach, while other diagnostic strategies can be used [Grastien *et al.*, 2011]. Finally the object of this article is also to argue that a consistency-based approach, as opposed to simulation-based approaches presented above, allows for less precise models, i.e., models that may not predict the future state but merely give constraints on the state variables.

1.2 Our Approach

In this work we propose an approach for the diagnosis of hybrid systems based on consistency. We use the framework of Grastien *et al.* [2012] which formulates the diagnosis problem as a series of diagnosis questions, each of which tests the consistency of some diagnostic assumptions with the diagnosis problem at hand. We assume that the number of discrete steps can be bounded and we reduce each question to the problem of finding an assignment to the state variables (both discrete and continuous) at these time steps that is consistent with (i.e., does not contradict) the model, the observation, and the assumptions. More specifically we reduce the diagnosis question to a constraint satisfaction problem written in the SMT (SAT Modulo Theory) framework. As opposed to the classical approach where the state of the system at a given time is defined as a function of the state at previous time (time being either discrete or continuous), we define the model as a number of constraints between the

values of the continuous and discrete variables at the current and the previous (discrete) times. For simplicity we restrict ourselves to piece-wise linear constraints; our approach can deal with more general models, but the number of existing solvers for such SMT problems drops quickly with the complexity of the constraints.

Compared to the simulation-based approaches, our approach does not require to maintain a probabilistic belief state which, to be precise enough, is prohibitive to maintain (in the case of the Adapt Lite system used for our experiments, the discrete space is already in the order of 10^{12} without considering the real-valued state variables). Furthermore we can deal with unknown behavioural modes: state variables whose evolution in certain modes is unspecified; in such situations, the CSP simply leaves these state variables assignments free. Compared to the redundancy-based approaches, our approach can accomodate any observability.

The here-proposed approach bears some similarities with bounded model-checking for hybrid systems [Audemard *et al.*, 2004]. There are however significant differences in the type of SMT problems generated. BMC for instance is incomplete and, consequently, is generally used to its limits; with diagnosis, we want to return solutions that are as precise as the model allows. On the positive side, diagnosis is based on a partial-observed observed run of the system, which means that the state space that needs to be explored is more restricted than in a model-checking problem.

We first present the hybrid system model and the diagnostic problem. We next present the reduction to SMT of a diagnostic test, and illustrate the implementation on the Adapt-Lite from the DX Competition.

2 Hybrid Systems

Hybrid systems are a class of models that include both variables that evolve continuously and variables that evolve discretely. The evolution of the continuous variables is usually defined by a set of equations that changes according to the value of the discrete variables.

Ideally the evolution in the continuous space is described by differential equations: $\dot{\mathbf{x}} = f_Q(\mathbf{x}, \mathbf{i}, \mathbf{w})$ where $\mathbf{x}$ is the vector of (continuous) state variables, $\dot{\mathbf{x}}$ is the differential state vector, $\mathbf{i}$ is the input vector, $\mathbf{w}$ is a (usually white Gaussian) noise, and Q is the system mode (discrete state). For computational reasons complex differential equations are often dropped in favour of difference equations: $\mathbf{x}@\tau' = f_Q(\mathbf{x}@\tau, \mathbf{i}, \tau' - \tau, \mathbf{w})$ where $\tau < \tau'$ are two times such that the input and the mode of the system is unchanged during the time period $[\tau, \tau']$. The problem is often restricted to situations where f_Q is linear, which is an acceptable assumption when piece-wise linear approximations are precise enough (each piece then corresponds to a different system mode).

The models presented above are used either to simulate the system or to deduce useful indicators. In a consistency-based approach, we do not need such a precise model, although more precision in the model will mean more precision in the diagnosis. We will still consider a discretisation of time, i.e., that the state of the system will be specified at important instants called timesteps although the time of these timesteps will be unspecified in general, and their number unbounded. The model will simply be a collection of *constraints* between the values of system variables at consecutive timesteps. Notice that there is a reversal with discrete event systems where the timesteps are generally associated with transitions, and states are represented for time intervals.

To simplify notations, we assume that each state variables is defined over the set of reals $\mathbf{R}$; one such variable is `time` which models the time. We limit ourselves to linear constraints because the number of solvers and their performance diminish quickly with more general constraints. A linear expression is an inequality involving over linear terms, i.e., a linear combination of variables with rational coefficients. A linear constraint is a collections of linear expressions connected with the logical operators such as $\neg$, $\wedge$, and $\vee$.

Our definition of hybrid systems is very similar to the one proposed by de Moura et al. [2008] except that i) I does not refer to the initial state (which is included in the observations) but to the state invariants, ii) the state invariants are extracted from the transition relation, and iii) to simplify modeling, we distinguish *soft* invariants from *hard* invariants. Hard invariants are constraints that apply in any state; soft invariants can temporarily become true as an effect of an instantaneous transition but must lead to another instantaneous transition. For instance, a nominal circuit breaker cannot remain closed in a state where the current is over a given limit; however that limit can be temporarily exceeded as a consequence of a short circuit.

Definition 1 *A* hybrid system *is a tuple* $\langle V, I_{\mathrm{H}}, I_{\mathrm{S}}, T\rangle$ *where* V *is a set of* variables*,* I_{H} *and* I_{S} *are two sets of hard and soft* invariants *both defined as sets of constraints over* V*, and* T *is a set of* transition constraints *defined as a set of cosntraints over* $V \cup V'$ *where* V' *is a copy of the variables in* V*.*

The set of transition constraints is here interpreted as a conjunction of constraints; we could see them as a disjunction of constraints i.e., an enumeration of how the system can evolve, or better as a conjunction of disjunctions, i.e., a set of synchronised local models.

A *state* of the hybrid system $\langle V, I_{\mathrm{H}}, I_{\mathrm{S}}, T\rangle$ is an assignment of the state variables: $V \to \mathbf{R}$. A *run* is a sequence of states $s_0, \ldots, s_k$ such that:

- all states s_i satisfy the hard constraints where v refers to $s_i(v)$;
- a state s_i may not satisfy a soft constraint iff a discrete transition occurred before and after s_i, i.e., $s_{i-1}(\mathtt{time}) = s_i(\mathtt{time}) = s_{i+1}(\mathtt{time})$;
- all pairs of consecutive states s_i, s_{i+1} satisfy all transition constraints where a variable v refers to $s_i(v)$ and a variable v' refers to $s_{i+1}(v)$.

A continuous transition is a pair of consecutive states s_i, s_{i+1} such that $s_i(\mathtt{time}) < s_{i+1}(\mathtt{time})$. This transition is represented as follows $s_i \xrightarrow{\delta} s_{i+1}$ where $\delta = s_{i+1}(\mathtt{time}) - s_i(\mathtt{time})$. It is assumed that if such a transition exists, then for all $\delta' \in]0, \delta[$, there exists a state s' such that $s_i \xrightarrow{\delta'} s'$ and $s' \xrightarrow{\delta-\delta'} s_{i+1}$ exist too. This can be ensured by considering only convex constraints (the only Boolean operator allowed is $\wedge$).

3 Diagnostic Problem

A model-based diagnosis problem is defined by a model, some observations, and some faulty behaviours. We now

describe the missing elements of this definition.

3.1 Observations

The framework proposed in this paper is very flexible with respect to the observations: essentially, an observation is just an information about the system run. We illustrate state-based observations and event-based observations, but more complex observations would be allowed. In particular, it is assumed here that the observations are perfect; any uncertainty on the observations is modeled in the hybrid system itself.

Definition 2 (State-based observations) *The* state-based observations *are a collection of triples $\langle \tau, v, \nu \rangle$ where τ is the time of observation, v is the observed variable, and ν is the observed value.*

Notice that state-based observations do not assume that all observed variables are observed at the same time or at the same rate. The initial state can be modeled as (possibly partial) observations of the initial state.

A run is consistent with the state-based observation $\langle \tau, v, \nu \rangle$ if it contains a state s such that $s(\texttt{time}) = \tau$ and $s(v) = \nu$.

To keep the model simple, we did not explicitly represent events. Instead, we assume that a variable v models the occurrence of events such that certain discrete transitions set v to 1 and all other discrete transitions and all continuous transitions set v to 0.

Definition 3 (Event-based observations) *The* event-based observations *are a tuple $\langle \Sigma_o, O \rangle$ where Σ_o is a subset of* observable variables *modeling the occurrence of observable events and O is a set of pairs $\langle v, \tau \rangle$ where $v \in \Sigma_o$ is the observed event and τ is the time when the event was observed.*

As opposed to state-based observations, event-based observations provide information not only when the events are observed but also when they are not. A run $\rho = s_0, \ldots, s_k$ is consistent with the event-based observations $\langle \Sigma_o, O \rangle$ iff $obs_{\Sigma_o}(\rho) = O$ where $obs_{\Sigma_o} = \{\langle v, \tau \rangle \in \Sigma_o \times [s_0(\texttt{time}), s_k(\texttt{time})] \mid \exists i \in \{0, \ldots, k\}.\ s_i(v) = 1 \wedge s_i(\texttt{time}) = \tau\}$.

3.2 Faults

Diagnosis is about determining what is faulty in a system. A fault may be defined as a pattern of events [Jéron *et al.*, 2006]. More often though is a fault in dynamic systems defined as the occurrence of a special type of event (a "faulty" event). Equivalently a fault can be seen as the specific assignment of a state variable; this is the definition used in this document.

Definition 4 (Faults) *The faults are a set of variables $V_f \subseteq V$.*

The faulty state of a run $s_0, \ldots, s_k$ is the set of faulty variables that are assigned to 1 at the end of the run: $\{v \in V_f \mid s_k(v) = 1\}$.

3.3 Diagnosis

A diagnostic problem is defined by a tuple $\langle M, o, V_f \rangle$ where M is a hybrid system, o are observations, and V_f is the set of faults.

A diagnostic candidate is a subset of faults $F \subseteq V_f$ such that there exists a run allowed by the model, that can generate the observations, and whose faulty state is precisely F. The diagnosis is the set $\Delta \subseteq 2^{V_f}$ of diagnostic candidates. Because the diagnosis is often uncertain (i.e., the size $|\Delta|$ is large) the focus is often on the minimal candidates, i.e., the sets F such that $F' \subset F \Rightarrow F' \notin \Delta$. The minimal diagnosis is the subset of minimal diagnostic candidates.

The consistency-based approach to diagnosis has been formalized by Reiter [1987] and has been recently extended for dynamic systems [Grastien *et al.*, 2011]. Consistency-based algorithms work by iteratively testing the intersection of Δ with sets of diagnostic hypotheses. Such tests are implemented by checking the consistency of i) the model and the observations (implicitly representing the diagnosis) with ii) some assumption on the faulty state (representing the diagnostic hypotheses).

The present work follows this approach and the following section is therefore dedicated to answering such diagnostic questions.

4 Answering a Diagnostic Question

We consider the problem of deciding whether a set of assumptions on the faulty state is consistent with a hybrid system and its observations. In this paper, we reduce this problem to a SAT Modulo Theory (SMT) problem and to solve it using state of the art technology in SMT.

SMT is an extension of the problem of propositional satisfiability (SAT) to contain operations from various theories such as the Boolean, bit-vectors, arithmetic, arrays, and recursive datatypes [de Moura *et al.*, 2007]; the linear arithmetic ($\mathcal{LA}$) is sufficient for this paper.

A $\mathcal{LA}$ problem is a tuple $\langle \mathcal{V}_B, \mathcal{V}_L, Cs \rangle$ where $\mathcal{V}_B$ is a set of Boolean-valued variables, $\mathcal{V}_L$ is a set of real-valued variables, and $Cs \subset \text{Constraints}(\text{LI}(\mathcal{V}_L) \cup \mathcal{V}_B)$ is a set of constraints defined by the Boolean-valued variables and linear inequalities over the set of real-valued variables. A *solution* to a $\mathcal{LA}$ problem is an assignment of the variables $\mathcal{V}_B \cup \mathcal{V}_L$ that makes all the constraints in Cs logically true. The problem is said *satisfiable* if there exists a solution, *unsatisfiable* otherwise.

4.1 Defining the Set of Variables

The reduction of a diagnostic question to a $\mathcal{LA}$ problem bears many similarities with the reduction from classical AI planning to SAT [Kautz and Selman, 1996] and even more with Bounded Model Checking for hybrid systems [de Moura *et al.*, 2008]. The SMT solver will be asked to find a system run that generates the observations while satisfying the faulty assumptions. As usual, we assume that the number n of transitions in the system run is bounded. Since any continuous evolution can be splitted in any number of transitions, it can be assumed that the number of transitions is exactly n. We are therefore looking for the run $s_0, \ldots, s_k$ which can be modeled by defining for each variable $v \in V$ and each timestep $i \in \{0, \ldots, k\}$ an SMT variable $v@t$ which will be assigned the value ν iff $s_t(v) = \nu$.

4.2 Translating the Model

We define a set of SMT constraints whose set of solutions maps exactly the system runs with $k + 1$ states.

For every hard constraint C, for every timestep $t \in \{0, \ldots, k\}$, we include the constraint $C@t$ that corresponds to the constraint C where every variable v is replaced by $v@t$.

Similarly a soft constraint must be satisfied in a state unless the state is surrounded by discrete transitions: for every soft constraint C, for every timestep $t \in \{0, \ldots, k\}$, we include $C@t \vee (\texttt{time}@(t-1) = \texttt{time}@t \wedge \texttt{time}@t = \texttt{time}@(t+1))$ (whether the soft constraints should be satisfied in the initial/final states is debatable).

Because the set of transition constraints is interpreted as a conjunction, each such constraint must be satisfied. We write $C@(t, t')$ the rewritting of the constraint C where each variable v is replaced by $v@t$ and each variable v' is replaced by $v@t'$. Each transition constraint C is therefore enforced by the SMT constraints $C@(t-1, t)$ where $t \in [1, \ldots, k]$.

4.3 Translating the Observation

Given observations, we define a set of SMT constraints whose set of solutions maps the runs that generate these observations.

Regardless of the type of observations, we assume that a timestep t_τ is associated with each time τ mentionned in the observations.

Given the state-based observation $\langle \tau, v, \nu \rangle$, we simply enforce the SMT constraint $v@t_\tau = \nu$.

Given the event-based observations $\langle \Sigma_o, \{\langle \tau_i, v \rangle\}_{i \in [1,\ldots,n]} \rangle$, for every observable event $v \in \Sigma_o$ and every timestep $t \in \{0, \ldots, k\}$, if there exists $i \in [1, \ldots, n]$ such that $v_i = v$ and $t = t_{\tau_i}$, then we define the SMT constraint: $v@t = 1$; otherwise (if no such i exists), we enforce $v@t = 0$.

4.4 Translating the Assumptions

In the experiments next section, each test will propose to find a candidate F' that is strictly better ($F' \subset F$) than another candidate F already found. To do so, we need i) to disallow the faults in $V_f \setminus F$ and ii) to forbid all faults from F to be active:

$$\bigwedge_{v \in V_f \setminus F} v@k = 0 \ \wedge \ \bigvee_{v \in F} v@k = 0.$$

5 The Adapt System

The validation of this approach is based on the industrial track of the international diagnostic competition [Kurtoglu *et al.*, 2009], i.e., the Adapt-Lite EPS. The system is composed of roughly 35 components including 20 sensors (which may be subject to fault).

5.1 Modeling the Components

The object of this subsection is to illustrate how the components in Adapt can be modeled.

Real-Valued Sensors

Adapt includes a number of sensors. The real-valued sensors return the (real) value of a state variable (say `m`).

The sensor can be in nominal, offset, stuck, or drifting state. This is modeled by three faulty state variable `fo`, `fs`, and `fd`. The value actually observed is the value of the free variable `o`. In nominal state, the observation will be the actual value plus the noise (here limited to N), which is represented by the following hard constraint:

$$\texttt{fo} = 0 \wedge \texttt{fs} = 0 \wedge \texttt{fd} = 0 \Rightarrow \texttt{m} - N \leq \texttt{o} \leq \texttt{m} + N$$

Discrete variable `t` represents the offset:

$$\texttt{fo} = 1 \Rightarrow \texttt{m} - N \leq \texttt{o} - \texttt{t} \leq \texttt{m} + N.$$

Discrete variable `k` represents the stuck-at value.

$$\texttt{fs} = 1 \Rightarrow \texttt{o} = \texttt{k}.$$

Continuous variable `co` represents the current offset of a drifting, and is used similarly to the offset value. Contrary to `offset` though, its value varies continuously. The drift could be defined as a linear variation, i.e., $\texttt{co}' - \texttt{co} = \texttt{slope} \times (\texttt{time}' - \texttt{time})$, but this function is not linear. Therefore, either $(\texttt{time}' - \texttt{time})$ is assumed known for every timestep (which is a possibility), but in general the drift should be modeled in a discrete fashion as follows: $\texttt{fd} = x \Rightarrow m_1 \times (\texttt{time}' - \texttt{time}) \leq \texttt{co}' - \texttt{co} \leq m_2 \times (\texttt{time}' - \texttt{time})$ where m_1 and m_2 are the lower and upper bound of the x-th discretisation of the drifting.

Fan Output

The fan output is interesting because it varies very slowly (as opposed to the voltage for instance). Temperature (for light bulbs) acts similarly in the large Adapt system. The flow increases roughly linearly from 0 to a maximum M and decreases similarly depending. We model the flow with variable `v` and hard constraints are defined to forbid the variable to leave the interval $[0, M]$. Furthermore four continuous constraints define the continuous evolution such as:

$$(\texttt{running} \wedge \texttt{v} \leq M) \Rightarrow \texttt{v}' - \texttt{v} = sl \times (\texttt{time}' - \texttt{time})$$

where sl is the constant slope of the linear increasing function.

5.2 Experiments

We modeled the Adapt-Lite system proposed by the DX-Competition [Kurtoglu *et al.*, 2009]. We ran a series of experiments over diagnostic windows of 5 seconds each at a rate of 2 hertz. The initial state is assumed nominal.

The classical consistency-based approach to diagnosis [Reiter, 1987] tests whether the nominal state is consistent with the diagnostic problem and uses the conflict returned by the solver to generate more diagnostic tests until a solution is found. The `cvc4` solver used in our experiments does not implement the conflict generation as yet and we therefore turned to the so-called PLS strategy proposed by Grastien et al. [2011]. This strategy searches for any diagnosis (i.e., checks the consistency of trivial assumption with the model and the observation). The SMT solver returns a possible behaviour of the system. The diagnostic state is extracted from this behaviour and another consistency check is performed, this time with the assumption that the faulty state should be preferred.

We provide some experimental results on a few selected instances on Table 1. The experiments were performed on an Intel i5-2520M 2.5GHz with 3.75GiB and running GNU/Linux 2.6.43.8-1.fc15.x86_64. The solver used is cvc4-1.0-x86_64. Each diagnostic test is performed independently, i.e., not incrementally.

The runtime are still very weak, with generally 2 min to solve any problem. Notice however that we did not try to apply any optimization or heuristic. It is well known that the SMT encoding affects greatly the performance of the solver. Furthermore, the search strategy presented here might impair the performance: for instance, testing the consistency

instance	time	nb faults	nb tests
1	2mn20	0	11
2	2mn00	1	10
3	3mn00	1	9
4	0mn56	1	4
5	2mn15	2	11

Table 1: Experimental results: time: the time necessary to solve the diagnostic problem; nb faults: size of the minimal cardinality diagnosis returned by the diagnoser; nb tests: number of calls to the SMT solver.

of the nominal faulty state with the observations in the first instance takes less than 10 seconds, i.e., the conflict-based approach would solve the first instance in less than 10 seconds despite the current encoding's shortcomings.

An advantage of the PLS strategy is however that finding multiple cardinality minimal diagnoses is not harder than finding single cardinality diagnoses since the diagnoser can actually stop earlier when the minimal cardinality is larger. Considering the issue of multiple cardinality again, a simulation-based approach would struggle with a scenario involving several faults at the same time as the probability of any consistent scenario would be extremely low. Our approach, on the other hand, is able to deal with any number of faults.

6 Conclusion and Future Works

We presented a novel approach to diagnosis of hybrid systems that is based on consistency tests. We discussed its advantages compared to existing approaches. In short, it does not require the astronomical space necessary for simulation-based approaches while being highly flexible with respect to its input (model, observation, and fault).

The experimental results provided in this article show that the approach is feasible although it is not quite applicable yet. We are optimistic that careful reduction to SMT and dedicated implementation of the SMT solver can gain orders of magnitude in runtime. In particular, we believe that there is much more potential here than, say, in SMT-based solving of model-checking problems [Ábrahám *et al.*, 2005]. We believe, for instance, that observations (which are inexistant in model-checking) make the SMT problem much easier to solve; also, there is much more potential for decomposition of a diagnostic problem into problems of (potentially overlapping) subsystems.

Similarly to the redundancy-based approach, the consistency-based approach as presented here is short-sighted, in the sense that it does not take into account all observations since the beginning, but only the last bits of observation. This is one strength of the simulation-based approach which keeps in the belief state the important information about old observations. Typically for the Adapt System, one wants to remember the configuration of the network (which switches were open/closed). Incrementality, and in particular how much information should/needs to be remembered about past observations will allow to track faulty behaviours that manifest themselves in long periods.

Acknowledgments

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

[Ábrahám *et al.*, 2005] E. Ábrahám, B. Becker, F. Klaedtke, and M. Steffen. Optimizing bounded model checking for linear hybrid systems. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI-05)*, pages 396–412, 2005.

[Arulampalam *et al.*, 2002] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing (TSP)*, 50(2):174–188, 2002.

[Audemard *et al.*, 2004] Gi. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with MathSAT. In *Second International Workshop on Bounded Model Checking (BMC-04)*, pages 17–32, 2004.

[Benazera and Travé-Massuyès, 2009] E. Benazera and L. Travé-Massuyès. Set-theoretic estimation of hybrid system configurations. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 39(5):1277–1290, 2009.

[Blom and Bar-Shalom, 1988] H. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control (TAC)*, 33(8):780–783, 1988.

[de Moura *et al.*, 2007] L. de Moura, B. Dutertre, and N. Shankar. A tutorial on satisfiability modulo theories. In *Nineteenth International Conference on Computer-Aided Verification (CAV-07)*, pages 20–36, 2007.

[de Moura *et al.*, 2008] L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: from refutation to verification. In *20th International Conference on Computer-Aided Verification (CAV-08)*, pages 14–26, 2008.

[Ernits and Dearden, 2011] J. Ernits and R. Dearden. Towards diagnosis modulo theories. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 249–256, 2011.

[Grastien *et al.*, 2011] A. Grastien, P. Haslum, and S. Thiébaux. Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 60–67, 2011.

[Grastien *et al.*, 2012] A. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: theory and practice. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR-12)*, 2012.

[Heintz *et al.*, 2008] F. Heintz, M. Krysander, J. Roll, and E. Frisk. FlexDx: a reconfigurable diagnosis framework. In *Nineteenth International Workshop on Principles of Diagnosis (DX-08)*, pages 79–86, 2008.

[Hofbaur and Williams, 2004] M. Hofbaur and B. Williams. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 34(5):2178–2191, 2004.

[Jéron *et al.*, 2006] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier. Supervision patterns in discrete-event systems diagnosis. In *Seventeenth International Workshop on Principles of Diagnosis (DX-06)*, pages 117–124, 2006.

[Kautz and Selman, 1996] H. Kautz and B. Selman. Pushing the envelope : planning, propositional logic, and stochastic search. In *Thirteenth Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.

[Kurtoglu *et al.*, 2009] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman. First international diagnosis competition – DXC'09. In *20th International Workshop on Principles of Diagnosis (DX-09)*, pages 383–396, 2009.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)*, 32(1):57–95, 1987.

[Staroswiecki and Comtet-Varga, 2001] M. Staroswiecki and G. Comtet-Varga. Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems. *Automatica*, 37:687–699, 2001.

Diagnosing the Root Cause of Accidents in Autonomous Vehicle Environments

Yedidya Bar-Zeev[1] and **Meir Kalech**[1] and **Roni Stern**[2]
[1]Ben-Gurion University, Beer-Sheva, Israel
e-mail: {yedidya,kalech}@bgu.ac.il
[2]School of Engineering and Applied Sciences, Harvard University, MA USA
e-mail: {roni.stern}@gmail.com

Abstract

Even with an Autonomous Intersection Management (AIM) system, accidents may still occur if vehicles malfunction. We address the problem of identifying the vehicles that caused the accident. We formally define this problem as a model-based diagnosis (MBD) problem and propose a SAT-based and a conflict-directed approach to solve it. Using a novel conflict-detection algorithm, we show experimentally and theoretically that the conflict-directed approach is more efficient and scales better than the SAT-based approach.

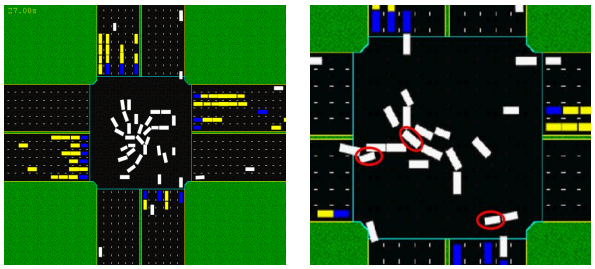
1 Introduction

Autonomous vehicles research stands in the front of the interests of vehicle companies. One challenging setting for autonomous vehicles is an intersection. While simple traversal rules such as traffic lights are possible, more advanced intersection management protocol can be much more efficient [VanMiddlesworth, Dresner, and Stone, 2008].

In particular, the Autonomous Intersection Management (AIM) project [Dresner and Stone, 2008] includes an efficient autonomous intersection management protocol for multiple autonomous vehicles. A vehicle wishing to pass through the intersection sends a request to the intersection manager (IM) and receives a path that is reserved for it for a specified time frame. If all agents follow the AIM protocol, it is guaranteed that no collisions will occur.

However, autonomous vehicles may malfunction, and as a result not follow their reservations. For instance, a communication error may cause an autonomous vehicle to arrive to the intersection more quickly than anticipated. Such a failure in a busy intersection with many autonomous vehicles may cause a chain accident, as shown in Figure 1(a). While safety measures [Dresner and Stone, 2008] can reduce significantly the probability of such an accidents from occurring, creating a fault-proof AIM system is challenging.

Assuming that accidents may occur, it is important to know which vehicles caused an accident, so these faulty vehicles can be fixed. We denote by $AIM\text{-}Diag$ the diagnosis problem of identifying these faulty vehicles, and address it in this paper. An example of a solution to an $AIM\text{-}Diag$ problem is shown in Figure 1(b), where the vehicles that caused the accident are marked in red circles.



(a) Complex accident scenario. (b) A diagnosis.

Figure 1: An example of an accident scenario.

One approach to solve $AIM\text{-}Diag$ is to install at the intersection hardware that will monitor the vehicles in the intersection accurately in real-time and identify which vehicle caused the accident. This approach goes beyond the basic AIM setting and is clearly more expensive to deploy. We propose an alternative, model-based diagnosis (MBD) approach that does not require any additional installations.

MBD relies on a model of the diagnosed system to compute the expected behavior of the system. When the expected and observed behavior differ, the model is used to infer possible failing components. In AIM, we use as a model the planned routes of the vehicles according to the IM. The observed behavior of the system is the location of the vehicles after the accident. Formalizing this problem in MBD terms is the first contribution of this paper.

The second contribution is adapting two MBD methods to solve $AIM\text{-}Diag$. The first method is based on translating $AIM\text{-}Diag$ to SAT and the second method uses a conflict-directed approach [de Kleer and Williams, 1987]. We show theoretically and experimentally that the SAT-based method does not scale. The efficiency of the conflict-directed method greatly depends on how it is able to detect *conflicts*. The third contribution of this paper is proposing an efficient conflict-finding algorithm that exploits the special properties of $AIM\text{-}Diag$.

Experiments on the AIM4 simulator [Stone et al.,] show that the proposed conflict-directed method is very fast, solving cases with hundreds of vehicles passing through the intersection and dozens of vehicles involved in the accident.

2 Diagnosis of a Reservation-Based AIM

In this section we define the $AIM\text{-}Diag$ problem and provide the relevant background. First, we briefly describe the reservation-based AIM [Dresner and Stone, 2008].

In the reservation-based AIM, the intersection is man-

aged by an Intersection Manager (IM) agent and every autonomous vehicle is controlled by a Driver Agent (DA). Let A be the set of DAs. Once a vehicle approaches an intersection, its DA sends a request to the IM, asking to pass through the intersection. Upon DA request, the IM plans a path in the intersection for the corresponding vehicle, checks if that path can be followed without colliding with another vehicle, and replies with an *accept* or *reject* response accordingly. An *accept* response includes the required acceleration and direction for each time step.

The IM plans paths and checks for collisions in the intersection by using a *reservation-based approach*. The intersection is divided into $n \times n$ tiles. When the IM accepts a request by a DA, it *reserves* the required set of tiles in the intersection. Note that since the vehicle will be moving through the intersection, the tiles reserved for it will be different for different time steps. Hence, the IM maintains for every relevant future time step t a $n \times n$ matrix that we call the *reservation matrix*.

Definition 1. [Reservation Matrix]. *The reservation matrix for time t, denoted by R^t is a $n \times n$ matrix, where*

$$r_{ij}^t = \begin{cases} a_k & \textit{the tile (i,j) is reserved for } a_k \in A \\ \emptyset & \textit{otherwise} \end{cases}$$

After an accident has occured, we can use the reservation matrices to restore the reserved path of each vehicle. The diagnosis methods described next use this information to infer which vehicles deviated from the planned reservation and caused the collision.

2.1 Problem Definition

The problem we address in this paper is how to discover the set of vehicles that have caused an accident in an AIM setting. The input to the problem is the observed location of the vehicles after the accident, and the set of reservation matrices, starting from some predefined time before the accident (regarded as time step zero) and until the time where the accident was identified and observed. To formally define this problem, which we call $AIM\text{-}Diag$, we introduce the following notation.

- $\mathbf{P}$: the set of all the tiles in the intersection.
- $\mathbf{t_{obs}}$: the time when the accident was observed.
- $\mathbf{O_i}$: the tiles used by a_i when the accident was observed.
- $\mathbf{R_i^t}$: the tiles reserved for agent a_i at time t.

Note that we do not assume that the accident is immediately identified as this would require additional hardware for identifying accidents in real time. Thus, t_{obs} represents a time step after the accident has occurred.

Definition 2. [$AIM\text{-}Diag$]. *Given the set of reservation matrices $\{R^t \mid t = 0, \ldots, t_{obs}\}$, and the observed location O_i for every agent $a_i \in A$, the $AIM\text{-}Diag$ problem is the problem of finding a set of DAs that have caused the accident.*

The rest of this paper will discuss methods to solve the $AIM\text{-}Diag$ diagnosis problem.

2.2 Diagnosis Approaches

There are diverse diagnosis approaches such as data-driven, case-based reasoning, knowledge-based systems and model-based diagnosis. Data-driven approaches are model-free statistical methods, that attempt to learn relations between the symptoms and the faults [Murray, Hughes, and Kreutz-Delgado, 2006]. These methods face the challenge of dimension reduction and a dependency on the existence of quality information that can be extracted from the data. Case-based reasoning approaches keep a repository of cases and their appropriate diagnosis. When a new case appears the case-based reasoning system tries to find the most similar case from the repository in order to adapt the relevant diagnosis [Aamodt and Plaza, 1994]. Both the data-driven and the case-based reasoning approaches do not guarantee sound diagnoses nor completeness.

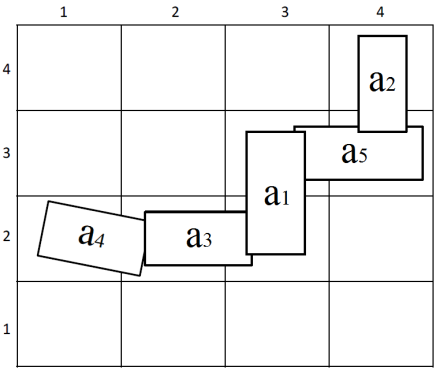


Figure 2: Observation matrix.

By contrast, model-based diagnosis (MBD) approaches are able to return a set of diagnoses that are sound and complete, in the sense that these diagnoses explain the observed behavior. MBD methods require a model of the diagnosed system, and use that model to infer the possible diagnoses. In this paper we adopt a MBD approach for $AIM\text{-}Diag$.

MBD have been previously proposed to cope with the diagnosis of general multi-agent systems [Jonge, Roos, and Witteveen, 2009]. The most closely related work is of Kalech and Kaminka [2007] that proposed MBD algorithms to diagnose coordination failures in MAS. The basic assumption in their work is that the coordination between the agents can be modelled in advance, while the coordination between the vehicles in $AIM\text{-}Diag$ is changed dynamically. Next, we describe the $AIM\text{-}Diag$ in MBD terms and propose two MBD approaches to solve it.

3 Model-Based Diagnosis for AIM

In MBD, a diagnosis problem is described by the triplet $\langle SD, OBS, COMPS \rangle$, where SD is a model of the diagnosed system, OBS is the observed behavior of the system and $COMPS$ is a list of system components.

To formalize $AIM\text{-}Diag$ as an MBD problem we introduce the $\mathbf{AT_{i,t,l}}$ and $LOC_{i,t,L}$ predicates. $AT_{i,t,l}$ is true if agent a_i occupied tile l at time t, and $LOC_{i,t,L}$ is true if a_i occupied at time t only the set of tiles L:

$$LOC_{i,t,L} = \bigwedge_{l \in L} AT_{i,t,l} \wedge \bigwedge_{l \in P \setminus L} \neg AT_{i,t,l}$$

We define the system components (COMPS) as the set of DAs, and the observations OBS as the set of tiles occupied by every agent at time t_{obs} (i.e., OBS_i for every a_i).

$$OBS = \bigwedge_{a_i \in A} LOC(i, t_{obs}, O_i) \qquad \text{(Observation)}$$

Consider, for example, the observation shown in Figure 2, assuming the observation time $t_{obs} = 6$. The corresponding observation clauses will be:

$$\begin{aligned} OBS \;=\; & LOC_{1,6,[(3,3),(3,2)]} \;\wedge\; LOC_{2,6,[(4,4),(4,3)]} \\ \wedge\; & LOC_{3,6,[(2,2),(2,2)]} \;\wedge\; LOC_{4,6,[(2,2),(1,2)]} \\ \wedge\; & LOC_{5,6,[(3,3),(4,3)]} \end{aligned}$$

3.1 Weak Fault Model

The basic model that is used by MBD algorithms, known as the Weak Fault Model (WFM), describes the behavior of every system component under the assumption that this component is healthy. The assumption that a component $c \in COMPS$ is healthy is denoted by $h(c)$, and the correct behavior of c is denoted by φ_c. Describing SD formally in a WFM is given by

$$SD = \bigwedge_{c \in COMPS} h(c) \rightarrow \varphi_c \tag{1}$$

In $AIM\text{-}Diag$, the correct behavior of a component (a driver agent) is to follow its reservations. Formally, for a component $c = a_i$ we can define φ_{a_i} as

$$\varphi_{a_i} = \bigwedge_{t \in \{1 \ldots t_{obs}\}} LOC_{i,t,R_i^t} \tag{2}$$

A diagnosis problem arises when $SD \wedge OBS$ in inconsistent with the assumption that all the agents are healthy. The output of an MBD algorithm is a set of *diagnoses*. A diagnosis is defined as a set of components Δ, such that assuming that they are faulty is consistent with $SD \wedge OBS$.

Definition 3. [Diagnosis] *Given an MBD problem,* $\langle SD, COMPS, OBS \rangle$*, the set of components* $\Delta \subseteq COMPS$ *is a* diagnosis *if*

$$SD \wedge \bigwedge_{c \in \Delta} \neg h(c) \;\wedge \bigwedge_{c \notin \Delta} h(c) \;\wedge\; OBS \;\nvdash \bot$$

For a given MBD problem, there may be many diagnoses that satisfies Definition 3. In MBD it is often preferred to search for *minimal* diagnoses, which are diagnoses that are not supersets of diagnoses. This corresponds to the assumption that a component is more probably to be healthy than faulty. In some cases, *minimal cardinality* (MC) diagnoses are preferred, which are the diagnoses with minimal amount of components assumed to be faulty. In $AIM\text{-}Diag$, searching for MC diagnoses is translated to seeking an explanation of the accidents that assumes the minimal amount of faulty driver agents. Using the above MBD formalization of SD, OBS and $COMPS$, one can theoretically use any off-the-shelf diagnosis algorithms [de Kleer and Williams, 1987; Williams and Ragno, 2007, inter alia] to find either minimal or MC diagnoses.

However, such WFM modeling is not sufficient to obtain helpful diagnoses in $AIM\text{-}Diag$. Consider a simple $AIM\text{-}Diag$ scenario with two agents a_1 and a_2, where a_2 diverged from its reserved tiles (due to some malfunction) and collided with a_1. If the observation time was exactly when the agents collided, then using SD and OBS as described above, one could infer that a_2 was not healthy, as it did not follow its expected reverberation. However, if the observation time was after the collision time, then the only possible diagnosis (which is consistent with Definition 3) is that both agents were faulty, since at time t_{obs} both agents were not in their reserved tiles: a_2 was not in its reserved tiles because it was faulty, and a_1 was not in its reserved tiles because it was blocked by a_2.

t	a_1	a_2	a_3	a_4	a_5
1	$(3,1)$	$(4,4)$			
2	$(3,1)$, $(3,2)$	**(4,4)**, **(4,3)**	$(1,2)$		
3	**(3,3)**, **(3,2)**	$(4,3)$, $(4,2)$	$(2,2)$, $(2,1)$	$(2,4)$	
4	$(3,4)$, $(3,3)$	$(4,2)$, $(4,1)$	**(3,2)**, **(2,2)**	$(2,4)$, $(2,3)$	$(4,3)$
5	$(3,4)$	$(4,1)$	$(4,2)$, $(3,2)$	$(2,2)$, $(2,3)$	**(3,3)**, **(4,3)**
6			$(4,2)$	$(2,1)$, $(2,2)$	$(2,3)$, $(3,3)$
7				$(2,1)$	$(2,3)$, $(1,3)$
8					$(1,3)$

Table 1: An example of a reservation matrix.

This example demonstrates that the WFM modeling described above is not sufficient to return meaningful diagnoses. Next, we resolve this problem by proposing an improved MBD modeling. In particular, we modify the system description (SD) described in (1) and (2) to include the behavior of the agents in the case of an accident. Modeling the faulty behavior of components is known in the MBD literature as using a strong fault model (SFM).

3.2 Strong Fault Model

The key notion that the following model attempts to capture is the behavior of healthy driver agents in the presence of faulty ones. This is done by replacing the previous definition of φ_c (Equation 2) with a set of constraints, which are described next.

The first constraint, denoted as *Constraint 1*, is that a healthy agent is at time $t + 1$ at its reserved location if at time t it was also in its reserved location and it did not collide with any other agent. To define Constraint 1 formally we introduce the predicate $Clear(i, t)$, which is true if the tiles reserved for agent a_i are not used by any other agent at time t.

$$Clear(i,t) = \bigwedge_{l \in R_t^i, a_j \in A} (i \neq j) \rightarrow \neg AT(j,l,t)$$

Using the $Clear$ predicate, Constraint 1 is defined for every agent a_i and time step $t < t_{obs}$ as follows:

$$(Clear(i,t) \wedge LOC(i,t,R_i^t)) \rightarrow LOC(i,t+1,R_i^{t+1})$$

(Constraint 1)

For example, if the reserved tiles for a_1 are $[(3,1),(3,2)]$ and $[(3,3),(3,2)]$ at times 2 and 3 respectively, then the relevant clauses for constraint 1 are:

$$\begin{aligned} (\text{Clear}(1,2) \;\wedge\; & LOC(1,2,[(3,1),(3,2)])) \\ \rightarrow\; & LOC(1,3,[(3,3),(3,2)]) \end{aligned}$$

The second constraint, denoted as *Constraint 2*, defines the location of DAs after a collision. As the focus of this work is the diagnosis algorithms and not physical modeling of accident kinematics, we take the simplifying assumption that a colliding vehicles become stuck in their location. In real world setting, a more elaborate physical model will be used.

Formally, we define the following constraint for every time t, tile l and pair of agents i and j as follows:

$$(AT_{i,t,l} \wedge AT_{j,t,l} \wedge (i \neq j)) \rightarrow \bigwedge_{l' \in P} (AT_{i,t,l'} \leftrightarrow AT_{i,t+1,l'})$$

(Constraint 2)

Note that while Constraint 1 applies to healthy DAs, Constraint 2 should be applied to all the DAs, including faulty driver agents. Therefore, the entire system description SD in this SFM modeling is as follows:

$$SD = \bigwedge_{a_i \in A, t \in 1 \ldots t_{obs}-1} h(a_i) \rightarrow (Constraint1)$$
$$\wedge \bigwedge_{a_i, a_j \in A, t \in 1 \ldots t_{obs}-1, l \in P} (Constraint2)$$

(SFM SD Model)

Given the above MBD formalization of $AIM\text{-}Diag$, one can solve $AIM\text{-}Diag$ with standard MBD solvers.

3.3 MBD Solvers

A known MBD approach for finding diagnoses is to *compile* SD and OBS to a representation that allows fast inference of diagnoses. Two such popular representations are OBDDs [Torasso and Torta, 2003] and DNNF [Darwiche, 1998]. This compilation approach is attractive because inference of diagnoses can typically be done in time that is linear in the size of the compiled representation. However, there is no guarantee that the size of the compiled representation will not be exponential in the number of system components.

Other work proposes to compile MBD to a SAT problem [Metodi et al., 2012], and then use SAT or MAX-SAT solvers to find possible diagnoses. Applying this approach for $AIM\text{-}Diag$ is discussed later in this paper.

A different approach to MBD is to apply a combination of deterministic reasoning and search algorithms. One classic approach involves a two-stage process. First, it identifies conflict-sets, each of which includes at least one fault. Then, it applies a hitting-set algorithm to compute sets of multiple faults that explain the observation [de Kleer and Williams, 1987; Williams and Ragno, 2007]. These *conflict-directed* approaches tend to fail for large systems due to infeasible runtime or space requirements. Later in this paper we present a conflict-directed algorithm that is scalable and efficient for $AIM\text{-}Diag$.

4 SAT-Based Approach

The relation between MBD and SAT solvers is well known and has been presented by various MBD algorithms [Metodi et al., 2012]. In fact, one of the currently best-performing MBD algorithms [Metodi et al., 2012] is based on compilation of the MBD problem to SAT.

The standard encoding of an MBD problem to SAT, as introduced by Metodi et. al. [2012], simply compiles the clauses in SD and OBS to a SAT formula α=$SD \wedge OBS$. The variables of α in our SFM model are the health variables $h(\cdot)$ and the locations of the DAs before t_{obs} (the $AT_{i,t,l}$ predicates). A satisfying assignment of α represents a diagnosis, where $h(a_i)$=$false$ means that a_i is assumed to be faulty. In order to find only MC diagnoses, one can use a MAX-SAT solver to find assignments with minimal health variables ($h(\cdot)$) set to false [Metodi et al., 2012].

4.1 Complexity

The main limitation of the SAT-based approach to $AIM\text{-}Diag$ is the size of the α. Let C be the maximal number of tiles reserved for a single DA. Table 2 describes the number of clauses required to encode SD. The row "# clauses" lists the number of clauses needed to encode each constraint. The row "# instances" lists the number of times each constraint is added to α. Assuming that $C << |P|^2$, the total number of clauses required to encode SD is $O(|A|^2 \cdot t_{obs} \cdot |P|^2)$. Although current SAT solvers can solve instances with millions of clauses, this approach is not a scalable solution. For instance, consider a problem with 15 driver agents, an intersection matrix of 30×30 and time horizon (t_{obs}) of 50 time stamps, the expected number of clauses is 9,112,500,000. Furthermore, the analysis in Table 2 is given in clauses stated in propositional logic. Standard SAT solvers usually compile the input clauses to CNF, which causes the number of CNF clauses to be even larger. In our standard PC, the largest instance we were able to run without exceeding memory limitations were on very small size problems of up to 6 agents and an intersection size of 20×20.

	Constraint 1	Constraint 2
# instances	$\|A\| \cdot t_{obs}$	$\|A\|^2 \cdot \|P\|$
# clauses	$t_{obs} \cdot C$	$\|P\|$
Total	$\|A\| \cdot t_{obs} \cdot C$	$\|A\|^2 \cdot t_{obs} \cdot \|P\|^2$

Table 2: # Clauses in the SAT-based approach

Algorithm 1: Conflict Directed MBD

Output: $Diags$, a set of diagnoses
1 $Confs \leftarrow$ initial set of conflicts
2 $Diags \leftarrow \emptyset$
3 **foreach** *minimal hitting set* Δ *of* $Confs$ **do**
4 **if** Δ *is a consistent diagnosis* **then**
5 Add Δ to $Diags$
6 Add a new conflict to $Confs$

5 Conflict-Directed Approach

An alternative class of MBD algorithms use *conflicts* to find diagnoses. A *conflict* is an assumption about which components are healthy that is inconsistent with OBS.

Definition 4. [Conflict]. *A conflict is a set* $\Gamma \subseteq COMPS$ *such that*

$$SD \wedge \bigwedge_{c \in \Gamma} h(c) \quad \wedge OBS \vdash \bot$$

It has been shown that diagnoses are hitting-sets of conflicts [de Kleer and Williams, 1987]. Conflict-directed diagnosis algorithms are built on this observation, following, in general, the framework outlined by Williams and Rango [2007] and shown in Algorithm 1. First, a set of conflicts is found using a conflict detection algorithm (line 1) such as ATMS [de Kleer and Williams, 1987] or LTMS. Then, a minimal hitting set of these conflicts is found using a hitting set algorithm. If this hitting set (Δ) is found to be a consistent diagnosis (according to Definition 3) it is added to the set of diagnoses (line 5). Otherwise, a new conflict is generated (line 6) and added to the set of conflicts to hit, so that future iterations would not return Δ. This can be achieved, for example, by adding $COMPS \setminus \Delta$ as a conflict or using previously found diagnoses. Conflict-directed MBD algorithms vary in the implementation of this framework. If the initial set of conflicts (line 1) contains all minimal conflicts, then every minimal hitting set is guaranteed to be consistent [de Kleer and Williams, 1987]. This is used in the well-known GDE algorithm [de Kleer and Williams, 1987] but is very computationally intensive.

Algorithm 2: *AIM-Diag* Conflict detection

```
Conflicts ← ∅
for a_i ∈ A do
    if rt_i = −1 then
        Add {a_i} to Conflicts
    else if rt_i < min_{a_j ∈ N(a_i)} rt_j then
        Add {a_i} ∪ N(a_i) to Conflicts
return Conflicts
```

Next, we adapt this conflict-directed approach to *AIM-Diag*, and propose a novel, polynomial, algorithm for finding the conflicts in *AIM-Diag*. While we do not prove that all the conflicts found by this algorithm are minimal, in our experimental evaluation we were able to always find all minimal diagnoses without needing to add more conflicts in addition to this initial set of conflicts.

5.1 Finding Conflicts in *AIM-Diag*

To explain our conflict-finding algorithm we introduce the notions of the *collision graph* and the *reservation time*.

Definition 5. [Collision Graph] *The* collision graph $CG = (V, E)$ *is an undirected graph in which every driver agent is a vertex, and a pair of agents a_i, a_j has an edge between them iff they share a tile in OBS, i.e. if they collided.*

As an example of a collision graph, consider the accident shown in Figure 2. The edges of the corresponding collision graph are (a_2, a_5), (a_5, a_1), (a_1, a_3) and (a_3, a_4). We denote by $N(a_i)$ the neighbors of a_i in CG.

Since the AIM is designed to avoid collisions, every connected component in CG is a conflict, as it represents an accident. However, these conflicts may be large, and smaller conflicts result, in general, a more efficient search for diagnoses. Next, we introduce the *reservation time*, which enables finding smaller, more informative, conflicts.

Definition 6. [Reservation Time] *The* reservation time *of an agent a_i, denoted by rt_i, is the earliest time according to the reservation table in which a_i was planned to reach its location in OBS (denoted by O_i). If O_i was never reserved for a_i we set $rt_i = -1$.*

It is easy to see that $rt_i = -1$ entails that a_i must be faulty, and hence $\{a_i\}$ is a conflict. Furthermore, consider from all the agents that are involved in a collision (i.e., agents with neighbors in CG) the agent a_i with the smallest reservation time. There are two options: either a_i was faulty or not. If a_i was not faulty, it arrived at O_i (its location in the observation time) exactly at time rt_i and collided with a member of $N(a_i)$. That member of $N(a_i)$ was therefore faulty, since a_i has the smallest reservation time among all the agents in $N(a_i)$. Thus, either a_i is faulty or one of its neighbors. Thus, the set $\{a_i\} \cup N(a_i)$ is a conflict. This notion is generalized beyond the single agent that has the smallest reservation time, as follows.

Lemma 1. *For any agent a_i, if $|N(a_i)| > 0$ and $rt_i < \min_{a_j \in N(a_i)} rt_j$ then $\{a_i\} \cup N(a_i)$ is a conflict.*

Proof. If $a_i \in V$ then either a_i was faulty or not. If a_i was faulty, then clearly $\{a_i\} \cup N(a_i)$ should be considered a conflict due to a_i.Assume that a_i is not faulty. This means that a_i has arrived at O_i at exactly time rt_i. Thus, agent a_i was involved in a collision at time rt_i with at least one of the agents in $N(a_i)$. Let a_j denote that agent. Since $rt_i < rt_j$, this means that a_j had to reach the collision point before its reserved time rt_j, and thus a_j was faulty.□

Algorithm 2 uses Lemma 1 to find conflicts in a computationally efficient manner, requiring at most $O(|A|^2)$, which is reasonable for very large number of agents. Then their minimal hitting sets are returned as diagnoses (Algorithm 1).

For example, the reservations time of the agents according to the reservation matrix shown in Table 1 are: $rt_1 = 3$, $rt_2 = 2$, $rt_3 = 4$, $rt_4 = -1$ and $rt_5 = 5$. Using Algorithm 2 we can generate the conflicts: $\{a_2, a_5\}$, $\{a_5, a_1, a_3\}$ and $\{a_4\}$. The minimal hitting sets of these conflicts are $\{a_4, a_5\}$, $\{a_1, a_2, a_4\}$ and $\{a_3, a_2, a_4\}$, which are indeed the diagnoses of this example.

6 Evaluation

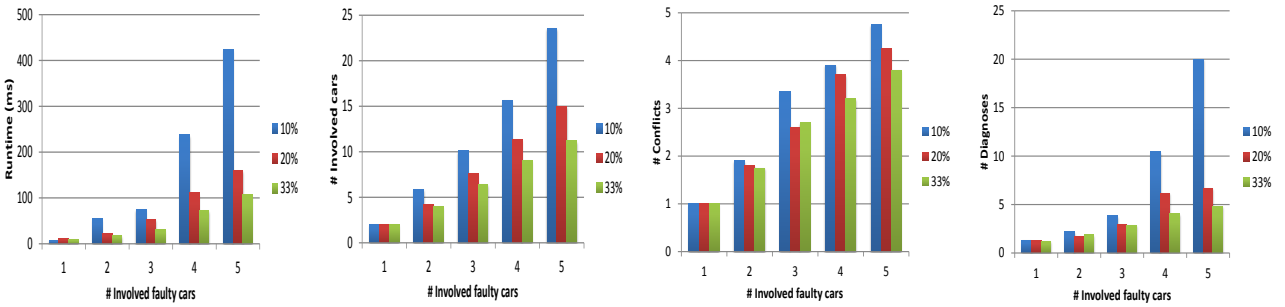
We evaluated the performance of the proposed conflict-directed diagnosis algorithm on top of the AIM4 simulator [Stone et al.,]. The AIM4 simulator simulates an automatic intersection, using the reservation-based communication protocol described earlier. In our experiments, we modified the AIM4 simulator to allow faulty cars to behave differently from the reservation. Faulty cars were generated according to a *fault rate* (FR) parameter, which dictates the ratio of normal to faulty cars. Faulty cars may drive slower or faster than expected (according to the reservation table) or drive outside of their reserved route. Thus, a faulty car may or may not cause an accident. For example, if the intersection is empty, then even if the car drives more slowly than expected, no accident will occur. When an accident occurs, we assumed that the cars involved in the accident simply stay in their current position.[1] Furthermore, we assume that the intersection management system is not immediately aware of the accident. As a result, more cars become involved in the accident when they collide into the cars that are already involved in the accident. The experiment was halted when the number of *faulty cars involved* (FCI) in the accident exceeded a predefined parameter.

We performed our experiments on an intersection with six lanes in every direction, represented by a grid of 60×60 tiles. We experimented with FR values of {10%, 20%, 33%} and FCI values of 1-5. For every scenario we run our conflict-directed method, until all minimal diagnoses were returned.

Figure 3(c) shows the average number of conflicts generated by our conflict generation algorithm. Figure 3(d), shows the average number of diagnoses returned. The x-axis of both figures is the FCI, the y-axis shows the average number of conflicts/diagnoses. First, as could be expected, we can see that the number of conflicts and the number of diagnoses are affected in a similar manner by the FCI and FR parameters: increasing the FCI or decreasing the FR resulted in an increase in both conflicts and diagnoses. This correlation between the number of conflicts and diagnoses is clear, since diagnoses are hitting-sets of conflicts.

To explain the effect of the FCI and FR on the number of conflicts, consider Figure 3(b), which shows in the y-axis the total number of cars involved in the accident. This includes both the faulty cars, that potentially caused the accident, as well as the other cars that were caught in the acci-

[1] In a real setting, one would need to implementing a more accurate physical model to account for the kinematics of the accident.



(a) Runtime in ms (b) Involved cars (faulty and not) (c) # of conflicts found (d) # of diagnoses returned

Figure 3: Experimental results for the conflict directed diagnosis method.

dent because of the faulty behavior of the faulty cars. Naturally, the number of involved **faulty cars** (FCI) is smaller than or equal to the total number of **cars** involved in the accident, and the number of involved cars will grow with the number of involved faulty cars. By contrast, a lower FR entails a larger number of involved cars, as this means that more non-faulty cars also pass through the intersection, with a higher chance to be "tangled up" in the accident. Thus, for the same FCI, having a lower FR would result in a higher number of involved cars. Therefore, increasing the FCI or decreasing the FR results in a larger number of involved cars, and as a result, a larger number of conflicts and diagnoses. Naturally, this results in higher runtime, as can be seen in Figure 3(a), which shows exactly the same trends as the number of conflicts and diagnoses shows.

We have also performed experiments using the proposed SAT-based method, using the SAT4J solver [Le Berre,]. Runtime results were substantially worse than with the conflict-directed approach. However, the main limitation was memory. Even for a scenario of five cars the average required memory was 2.5 GB. For six cars 4.2 GB were required, for seven cars 5.9 GB were required and for eight cars the average required memory was about 10.6 GB. Obviously this algorithm is not feasible even for small systems.

7 Summary

This paper addresses the problem of diagnosing car accidents in an autonomous vehicle setting. This problem was formalized as an MBD problem, where the model is the reservation table of the AIM and the observations are the state of the intersection after the accident. Two methods were described, the first reduces the problem into the MAX-SAT problem, allowing the use of MAX-SAT solvers to find the desired diagnoses. An alternative, and more scalable method uses a conflict-directed approach. Conflicts are identified using a specific algorithm for this setting, and diagnoses are extracted from them in a standard manner. Experimental results on top of the AIM simulator suggested the applicability of the conflict-directed method and discussed to some extent the attributes that affect its' performance.

References

[Aamodt and Plaza, 1994] Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.* 7(1):39–59.

[Darwiche, 1998] Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

[de Kleer and Williams, 1987] de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

[Dresner and Stone, 2008] Dresner, K. M., and Stone, P. 2008. Mitigating catastrophic failure at intersections of autonomous vehicles. In *AAMAS (3)*, 1393–1396.

[Jonge, Roos, and Witteveen, 2009] Jonge, F.; Roos, N.; and Witteveen, C. 2009. Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems* 18(2):267–294.

[Kalech and Kaminka, 2007] Kalech, M., and Kaminka, G. A. 2007. On the design of coordinated diagnosis algorithms for teams of situated agents. *Artificial Intelligence* 171:491–513.

[Le Berre,] Le Berre, D. Sat4j: a reasoning engine in Java based on the SATisfiability problem (SAT). `http://www.sat4j.org`.

[Metodi et al., 2012] Metodi, A.; Stern, R.; Kalech, M.; and Codish, M. 2012. Compiling model-based diagnosis to boolean satisfaction. In *AAAI*.

[Murray, Hughes, and Kreutz-Delgado, 2006] Murray, J.; Hughes, G.; and Kreutz-Delgado, K. 2006. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research* 6(1):783.

[Stone et al.,] Stone, P.; Dresner, K.; VanMiddlesworth, M.; and Au, T.-C. AIM4 simulator. `http://www.cs.utexas.edu/~aim`.

[Torasso and Torta, 2003] Torasso, P., and Torta, G. 2003. Computing minimum-cardinality diagnoses using OBDDs. *Advances in Artificial Intelligence (lecture notes in artificial intelligence)* 2281:224–238.

[VanMiddlesworth, Dresner, and Stone, 2008] VanMiddlesworth, M.; Dresner, K.; and Stone, P. 2008. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *AAMAS Workshop on Agents in Traffic and Transportation*, 94–101.

[Williams and Ragno, 2007] Williams, B. C., and Ragno, R. J. 2007. Conflict-directed A* and its role in model-based embedded systems. *Discrete Appl. Math.* 155(12):1562–1595.

Test Oracle Placement in Spectrum-based Fault Localization

Claudio Landi[1] and **Arjan van Gemund**[2] and **Marina Zanella**[1]
[1] University of Brescia, Italy
e-mail: {marina.zanella}@ing.unibs.it
[2] Delft University of Technology, The Netherlands
e-mail: a.j.c.vangemund@gmail.com

Abstract

In previous work it has been shown that Spectrum-based Fault Localization (SFL), used in software, can also be applied to hardware, thus offering a unified approach to the diagnosis of embedded systems that does not require the costly and error-prone activity of behavioral component modeling. However, the typically low diagnostic performance of SFL when applied to hardware currently prohibits practical use, leading to the need for more observability within the system. Also in software the limited branch topology, causing low testability, often necessitates additional observability. In this paper we present simple algorithms that determine the best location to insert additional test oracles (monitors, probes, invariants), and study their impact on diagnostic performance improvement. It is shown that even simple algorithms can considerably improve SFL's performance.

1 Introduction

Recent experience with applying Model-Based Diagnosis (MBD) in Dutch industry has shown that modeling cost is a bottleneck for the acceptance of MBD. At ASML (the world's leading lithography machine vendor) a LYDIA-based diagnoser has been installed on over 3,000 service laptops worldwide, and has successfully been used to diagnose faults in an important electro-mechanical subsystem that frequently suffered from failures [Pietersma and van Gemund, 2007]. It was shown that MBD can reduce solution cost from days to minutes for a one-only investment of 25 man-days of modeling effort (approximately 2,000 LOC, comprising sensor modeling, electrical circuits, and some simple mechanics). Despite the obvious financial gains, management discontinued the project once it become clear that only some 80% of the model could be obtained *automatically* from the system source codes (graphical schematics data for electrical, VHDL for the logic circuits) [Reeven, 2011; Schoemaker, 2008].

More recently, similar, disappointing experience has been obtained with an MBD project for the European Space Agency (ESA), which involved modeling the electrical power subsystem of the GOCE satellite [Feldman *et al.*, 2013]. Despite successful conclusion of this proof-of-concept research project, it appears to be highly uncertain whether the results will ever be applied in a follow-up project by the target ESA satellite operators, now that ESA has fully become aware of the complexity and human effort involved in the associated diagnostic satellite modeling.

In industrial situations where software and hardware is constantly evolving, there is growing evidence that mainstream adoption of MBD is only feasible when modeling can be *fully automated*. Given the recent success of SFL in the software domain [Abreu and van Gemund, 2010; Mathijssen, 2008; Zoeteweij *et al.*, 2007; 2008] it has recently been studied to what extent (model-based) SFL can offer an alternative to MBD in the logic hardware domain [van Gemund *et al.*, 2011]. In SFL the dynamic program execution profiles of tests are correlated with the test outcomes (pass/fail), typically by using statistical similarity coefficients. The components are subsequently ranked in order of the likelihood that they are responsible for test failures. As the spectra are captured by automatic profiling, and as the test oracles are readily implemented from existing specifications, *no* modeling effort is required.

In [van Gemund *et al.*, 2011] it has been shown that for specific ISCAS85 circuits SFL is capable to approach the diagnostic performance of MBD, in particular under a restrictive, single-fault assumption. Despite this limited success, at the same time it has become clear that in practice (i.e., multiple-fault) application of SFL to hardware, or coarse-grain software systems, is only feasible when observability is improved. This is because even for a test suite that exercises all possible execution paths, the information content of the spectra is too low for acceptable diagnostic performance, due to what is known as the *ambiguity problem*. Increasing observability implies increasing the number of test oracles (sensors) within the system under study. In software this would amount to inserting (more) invariants at specific points, while in hardware this is similar to inserting (more) probes [de Kleer and Williams, 1987]. In software it has already been shown that inserting invariants (also called screeners) at the statement level has a profound, positive impact on the diagnostic performance of SFL [Abreu *et al.*, 2008].

In this paper we study the improvement of SFL's diagnostic performance when placing additional test oracles, gener-

ically called *monitors*. In particular,

- We present an algorithm, coined Lion, that determines the next-best monitor placement, based on the static topology of the system under study. Monitor placement using Lion is typically applied iteratively offering an any-time solution. The typical application is in a static, design-time context, where it must be determined at which places to design in monitors (design for diagnosability).
- We present an algorithm, coined Tiger, that determines the next-best monitor placement, based on the topology of the system under study, as well as on the diagnostic results of a given test suite. Monitor placement using Tiger is typically applied iteratively offering an any-time solution. The typical application is in a dynamic, diagnosis-time context, where it must be determined at which place to add monitors to improve the current diagnosis.
- We evaluate the diagnostic performance improvement of both algorithms for increasing number of monitors based on an example embedded system simulator, taken from a real case study at Thales Naval Systems [Piel *et al.*, 2012].

To the best of our knowledge, these are the first model-free, monitor placement algorithms that are introduced in the context of SFL. While in the dynamic case, probing approaches have been proposed [de Kleer and Williams, 1987; Voas, 1996], these have been addressed in the context of MBD (thus using models), or are based on explicit, (software) system error propagation measurement, respectively. Furthermore, in [Brodie *et al.*, 2003] the term probing applies to active testing rather than monitor placement.

Our case study shows that the off-line run of Lion considerably improves SFL's performance, compared to random placement, at a time complexity of $O(N \cdot M^3)$, N being the number of tests and M the number of components. While Lion cannot exploit test execution results, Tiger benefits from its integration in the diagnostic loop. Even with its low-cost heuristic, Tiger is capable of approaching optimum placement performance within a factor close to 3 in terms of the number of monitors required.

The paper is organized as follows. In Section 2 we briefly introduce SFL and the associated ambiguity problem that we address in this paper. In Section 3 we present our algorithms, while in Section 4 we present our experimental results. In Section 5 we summarize our findings.

2 SFL

In this section we briefly introduce SFL, and show how adding additional monitors improves diagnostic performance. More details on SFL can be found in [Abreu *et al.*, 2007; Abreu and van Gemund, 2010; van Gemund *et al.*, 2011].

2.1 Modus Operandi

In SFL the following is given:

- A finite set $C = \{c_1, \ldots, c_j, \ldots, c_M\}$ of M components of which M_f are faulted.
- A finite set $T = \{t_1, \ldots, t_i, \ldots, t_N\}$ of N tests with binary outcomes $O = (o_1, \ldots, o_i, \ldots, o_N)$, where $o_i = 1$ if test t_i failed, and $o_i = 0$ otherwise.
- A $N \times M$ (test) coverage matrix, $A = [a_{ij}]$, where $a_{ij} = 1$ if test t_i involves component c_j, and 0 otherwise. Each row is also called a *spectrum* (hence the name SFL).

The health of component c_j is denoted $h_j \in \mathbb{R}$ where $h_j = 1$ represents full health, $h_j = 0$ represents faulted in all circumstances, while $0 < h_j < 1$ represents the case where c_j returns a failure in fraction h_j of the cases. In software this fraction h_j is related to *coincidental correctness* [Wang *et al.*, 2009] and *failure exposing potential* [Rothermel *et al.*, 2001], while in hardware this fraction is related to *failure intermittency* [de Kleer, 2007]. In our experiments we will inject faults with varying h_j.

The diagnostic result of SFL is a *component ranking* $R =< c_{r(1)}, \ldots, c_{r(j)}, \ldots, c_{r(M)} >$, ordered in terms of the likelihood $\mathsf{Pr}(c_j)$ that c_j is at fault. In a reasoning approach to SFL [Abreu and van Gemund, 2010] the $\mathsf{Pr}(c_j)$ are posteriors based on Bayesian probability theory. In statistical approaches to SFL the $\mathsf{Pr}(c_j)$ are approximated using statistical similarity coefficients [Abreu *et al.*, 2009].

For the purpose of this paper we choose similarity coefficients, in particular the Ochiai [Abreu *et al.*, 2007] coefficient, which is defined as

$$s_j = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{01}(j))(n_{11}(j) + n_{10}(j))}}$$

where $n_{pq}(j)$ is given by

$$n_{pq}(j) = |\{i \ : \ a_{ij} = p \wedge o_i = q\}|$$

For instance $n_{11}(j)$ is the number of occurrences where a test involving c_j returns a failure. The similarity coefficient is used as ranking criterion, i.e., we equate $\mathsf{Pr}(c_j) = s_j$.

2.2 Performance Metric

The diagnostic utility (accuracy, performance) of R is measured in terms of the identification cost C_d, which models the verification effort of a diagnostician, going down the suspect ranking R searching for the actual faults (true positives). In particular, we measure the identification effort *wasted* on false positives (i.e., excluding the components found to be faulted) in order to obtain a metric that is independent of M_f. Let r denote the index in R of the faulted component with the lowest index. Then the identification cost for all M_f faults equals $r-1$. In our studies we will typically consider a normalized value $C_d = (r-1)/(M-M_f)$ which ranges from 0 to 1 (a value of 0 indicates no identification effort, i.e., all faulted components are ranked at the top, whereas a value of 1 indicates maximum identification effort, i.e., all faulted components are at the bottom of the list). This normalized metric is essentially the inverse of the DXC utility metric [Feldman *et al.*, 2010] for diagnosers that produce no false negatives (R includes all components so it cannot miss any faulted component).

2.3 Ambiguity Reduction

While for fine-grain systems (e.g., software at the statement level) the high variability of test paths allows good diagnostic performance, for coarse-grain systems, such as hardware circuits or software systems profiled at the task level only (such as systems of systems), the variability of test paths is low, which gives rise to the so-called *ambiguity problem*. Consider the circuit shown in Fig. 1. Suppose that there is only one test possible that involves all 4 components, as

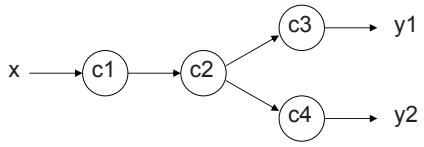


Figure 1: Example system

would be the case if the system were a hardware circuit [1]. When we apply SFL to this system each primary output observation is interpreted as one test. Thus one test vector (x) represents two tests, one involving c_1, c_2, and c_3 (the cone of the top primary output y_1), and one involving c_1, c_2, and c_4 (the bottom cone of y_2). In the above circuit it is assumed that y_1 and y_2 are readily monitored by a test oracle (or two oracles, depending on the format of the test specification).

For the sake of this example, assume that c_1 is at fault, and that we observe y_1 correct [2] (pass), and y_2 incorrect (fail). In terms of A and O we have

1	1	1	0	+
1	1	0	1	-

where '+' and '-' denote $o_i = 0$ (pass) and $o_i = 1$ (fail), respectively, to distinguish O from A in the figure. The associated similarity coefficients for $c_1, \ldots, c_4$ are given by $0.71, 0.71, 0, 1$, respectively. Consequently, c_4 is deemed most suspect, while c_3 is entirely exonerated. In terms of diagnostic performance we have $C_d = 0.5$ since, after c_4 is inspected, half of the times the diagnostician will hit on c_1. SFL's disappointing diagnostic performance ($C_d = 0.5$, no better than random) is due to the *ambiguity group* (c_1, c_2), that have equal columns in A and therefore end up in the ranking with equal similarity coefficient.

As for this system no other tests are possible, in order to improve the diagnosis we need to add another monitor to the system, since additional tests would not change the spectrum for this topology. Suppose we choose the output of c_1 to place the new monitor in order to break the ambiguity group. This is shown in Figure 2. If we run the same test we

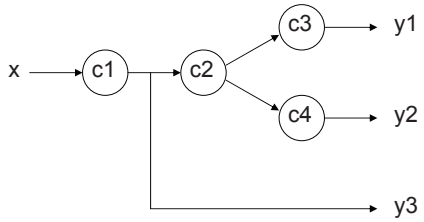


Figure 2: Example system extended with additional monitor

now obtain

1	1	1	0	+
1	1	0	1	-
1	0	0	0	-

where the third row in A, due to the additional monitor, breaks the ambiguity group. In the above example the output of c_1 (y_3) turns out to be incorrect. The associated similarity coefficients for $c_1, \ldots, c_4$ are now given by $0.82, 0.50, 0, 0.71$, respectively. Consequently, c_1 is now ranked on top and $C_d = 0$, i.e., perfect diagnostic performance.

3 Algorithms

In this section we present our monitor placement algorithms Lion and Tiger. Both algorithms are based on a simple, myopic heuristic, and return a monitor placement that approaches the next-best placement. The algorithms are intended to be called successively, until sufficient diagnosability is reached, as explained in the following sections.

3.1 Lion

The Lion algorithm is intended for a static, design context, where a system topology and associated test suite are available (i.e., a coverage matrix A), but where the tests are not yet executed (or executable), such as in a design for SFL-diagnosability context.

As the ambiguity of A is the target to be minimized, Lion searches for the largest ambiguity group in A, denoted C' (a subset of C). If one exists, it adds a monitor at a location that minimizes its ambiguity.

Let $P \subseteq C$ denote the subset of components whose output is (already) monitored by a test oracle. In a hardware circuit this initially corresponds to each primary circuit output (hence the name P). In Fig. 1 $P = \{c_3, c_4\}$. In software initially $|P| = 1$ as the primary output of the program is the program's return value(s). In case of multiple program outputs these are considered as one (component) output since there is only one test oracle (per test).

A sketch of the essentials of the algorithm appears in Algorithm 1 (the actual algorithm can be found in [Landi, 2012]). Function GETLARGESTAG returns the largest ambiguity group (AG) from the components selected as parameter (initially C). If there exist more than one AG, one of them is randomly chosen. Function UPDATEA adds the new rows to A that are generated by those tests in the test suite T that exercise c (cf. the example in Section 2.3). The algorithm selects the c that minimizes A's new ambiguity group size (s), and adds the new component to the set of components P that are monitored by test oracles. Note that each time a monitor (component) is added, it is possible that other ambiguity groups are also split, which produces a further ambiguity decrease. The algorithm returns the updated A and P, which can be used when Lion is recursively applied. The ambiguity size is typically used as termination criterion.

The time complexity of the algorithm is as follows. GETLARGESTAG takes $O(M^2 \cdot N)$ operations. The search for component c' costs $O(M)$ (size of C') times the matrix update ($O(N)$ partial row copies) plus identifying the largest AG ($O(M^2 \cdot N)$), which totals $O(N \cdot M^3)$. The space complexity of the algorithm is negligible compared with the space complexity of the coverage matrix.

3.2 Tiger

The Tiger algorithm is intended for a dynamic context where test outcomes are available, such as an operational context (run-time tests, e.g. invariants).

[1] Many (software) systems will admit tests that only exercise component subsets, which is attractive from SFL point of view. However, in this example we choose to ignore this for the purpose of the exposition.

[2] Note that a fault doesn't have to manifest itself at the component's output, nor propagate to a system output.

Algorithm 1 LION

```
function LION(A, P)
    C' ← GETLARGESTAG(A, C)
    if |C'| = 1 then
        return NULL
    end if
    s = |C'|
    for each c ∈ C', c ∉ P do
        A' ← UPDATEA(A, c)
        C'' ← GETLARGESTAG(A', C')
        if |C''| < s then
            s ← |C''|
            c' ← c
        end if
    end for
    A' ← UPDATEA(A, c')
    P' ← P ∪ {c'}
    return (A', P')
end function
```

Rather than aiming to minimize the ambiguity of A, Tiger exploits the additional test information, available in terms of the diagnosis R. The main idea is to assist SFL in minimizing the ambiguity of R (in probabilistic sense). To this end, Tiger selects the most suspect component that doesn't yet have a monitor. In the example in Section 2.3 this would be either c_1 or c_2 (whichever comes first starting from the top of R), since c_4 (most suspect) already is monitored.

A sketch of the essentials of the algorithm appears in Algorithm 2 (the actual algorithm, which is slightly more advanced, can be found in [Landi, 2012]). Function SFL computes the component ranking R given coverage matrix A and test outcomes O. After the new component c is selected for monitoring, Tiger recomputes the matrix, retests the system (function TEST), and returns the new A, P, O to allow recursive application.

Algorithm 2 TIGER

```
function TIGER(A, P, O)
    R ← SFL(A, O)
    R' ← R \ P
    if |R'| = 0 then
        return NULL
    end if
    c ← r'_1
    A' ← UPDATEA(A, c)
    O' ← TEST(T)
    P' ← P ∪ {c}
    return (A', P', O')
end function
```

As the search for a new monitor location is limited to finding the top component in R that doesn't yet have a monitor, the time complexity of Tiger is entirely determined by the SFL algorithm ($O(M(N + \log M))$, the matrix update ($O(N \cdot M)$), and executing the test suite (amounting to $O(N \cdot M)$). The space complexity of Tiger is negligible compared with the space complexity of the coverage matrix.

3.3 Running Example

Consider the system shown in Fig. 3. Assume that only components c_4 c_8 are initially monitored by a test oracles ($P = \{c_4, c_8\}$).

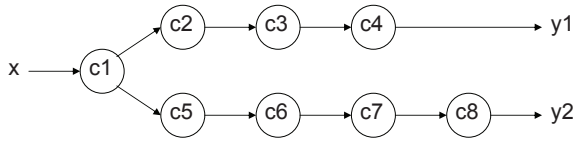


Figure 3: Example system

Execution of Lion

Lion can be iteratively invoked to place several additional monitors in the system of Fig. 3, so as to reduce the ambiguity of the associated coverage matrix. In the first invokation, Lion finds the set of all the ambiguity groups $AGs = \{\{c_1\}, \{c_2, c_3, c_4\}, \{c_5, c_6, c_7, c_8\}\}$. The largest ambiguity group is $LAG = \{c_5, c_6, c_7, c_8\}$. Lion chooses to place an additional monitor at the output of component c_6 to split LAG into two AGs to minimize their size (c_6 is added in P). Lion acts similarly in the next invokations, as shown in Table 1 [3], to place further monitors. In the third invokation there are several $LAGs$, and one of them is randomly chosen.

Lion places additional monitors off-line. Assume that, during the operating life of the system, component c_3 becomes faulty. Table 2 shows the ranking improvement obtained by increasing the number of the monitors according to Lion's suggestions. Note that three additional monitors (connected to c_6, c_2, and c_3, respectively) are required to obtain an optimal ranking, that is, a ranking such that the the faulty components are ranked higher than all healthy components.

Execution of Tiger

Again, consider the system shown in Fig. 3. Assume that component c_3 is at fault and Tiger is recursively invoked to place additional monitors in the system of Fig. 3, until an optimal ranking is achieved. In its first invokation, Tiger produces the ranking $R =< c_2, c_3, c_4, c_1, c_5, c_6, c_7, c_8 >$. Consequently, Tiger decides to place an additional monitor at the output of c_2, as this component is the first of R that is not monitored yet (c_2 is added in P). Tiger acts analogously in the subsequent invokations, as shown in Table 3 [3]. As expected, the ranking improves with the increasing number the monitors. Note that only two additional monitors (at c_2 and c_3) are sufficient to obtain an optimal ranking.

4 Experimental Results

The algorithms have been tested using a situations awareness system developed at Thales Naval Systems of which a smaller version was made available in the context of a research project [van der Laar *et al.*, 2013; Piel *et al.*, 2012]. Instead of using the real system (which involves a lot of proprietary naval hardware) the evaluation was performed using a simulator based on the real system, where we have effectively used the system execution graph (DAG) topology. The original DAG was modified to include some more components and random execution paths (tests) to make the diagnosis task more challenging. In terms of the DAG the system comprises $M = 74$ components. There are 10 test

[3] The numbers between brackets in column R (ranking) are the fault similarity coefficients of components. The value of the similarity coefficient of all missing components is zero.

call	P	AGs	LAG	add in P
1	$\{c_4, c_8\}$	$\{\{c_1\}, \{c_2, c_3, c_4\}, \{c_5, c_6, c_7, c_8\}\}$	$\{c_5, c_6, c_7, c_8\}$	c_6
2	$\{c_4, c_8, c_6\}$	$\{\{c_1\}, \{c_2, c_3, c_4\}, \{c_5, c_6\}, \{c_7, c_8\}\}$	$\{c_2, c_3, c_4\}$	c_2
3	$\{c_4, c_8, c_6, c_2\}$	$\{\{c_1\}, \{c_2\}, \{c_3, c_4\}, \{c_5, c_6\}, \{c_7, c_8\}\}$	$\{c_3, c_4\}$	c_3

Table 1: Lion invokations

P	R	C_d
$\{c_4, c_8\}$	$< c_2(1), c_3(1), c_4(1), c_1(0.70), c_5(0), \cdots >$	0.14
$\{c_4, c_8, c_6\}$	$< c_2(1), c_3(1), c_4(1), c_1(0.58), c_5(0), \cdots >$	0.14
$\{c_4, c_8, c_6, c_2\}$	$< c_3(1), c_4(1), c_2(0.71), c_1(0.5), c_5(0), \cdots >$	0
$\{c_4, c_8, c_6, c_2, c_3\}$	$< c_3(1), c_2(0,82), c_4(0.71), c_1(0.63), c_5(0), \cdots >$	0

Table 2: Rank improvement obtained using Lion

call	P	R	C_d	add in P
1	$\{c_4, c_8\}$	$< c_2(1), c_3(1), c_4(1), c_1(0,71), c_5(0), \cdots >$	0.14	c_2
2	$\{c_4, c_8, c_2\}$	$< c_3(1), c_4(1), c_2(0,71), c_1(0.58), c_5(0), \cdots >$	0	c_3
3	$\{c_4, c_8, c_2, c_3\}$	$< c_3(1), c_2(0.82), c_1(0.71), c_4(0.71), c_5(0), \cdots >$	0	c_1

Table 3: Tiger invokations

suites available, each providing $N = 33$ distinct execution paths. The performance of Lion and Tiger are measured in terms of the diagnostic performance metric C_d, obtained from injecting M_f faults, running the tests and performing diagnosis, while increasing the number of monitors as subsequently computed by either Lion or Tiger. In order to provide a reference, the algorithms are compared to Random monitor placement (upper bound on C_d) as well as Brute-Force placement (lower bound on C_d, enumerating over all $M - |P|$ monitor placements).

In the experiments each of the 10 test suites is executed, and the C_d results are averaged over the 10 test suites. Within each test suite the C_d results are averaged over random fault injections, executing each test multiple times to sample from the random faulty component health (when $h > 0$). In total, per test suite the number of tests executed is 450 per monitor placement. Due to the complexity of brute-force computation the experiments have been restricted to single fault injections only ($M_f = 1$).

Fig. 4 shows the results for an increasing number of additionally placed monitors (up to $M - 1 = 73$) where the first C_d value corresponds to the initial situation with $|P| = 1$ monitor. The fault injected has a health value of $h = 0.5$, i.e., the probability that the fault manifests itself at a monitor is 50%. The noise in the curves is due to the limited amount of samples used to average C_d per monitor placement. Brute-force placement yields near-perfect results even after a few monitors. Tiger takes slightly more than twice the number of monitors to achieve optimum performance. Both Lion and Random perform less, as in neither case the test outcomes are exploited. Nevertheless, Lion performs far better than Random, as the AG size in A is a good predictor of diagnostic performance. More experimental results can be found in [Landi, 2012].

5 Conclusion

While SFL offers the advantage of a model-free approach to diagnosis, in practice the inherently limited testability of many hardware and software systems severely limits SFL's diagnostic performance, compared to MBD. In this paper we present two algorithms (Lion and Tiger) that determine the best location to insert additional test oracles (monitors, probes, invariants), and study their impact on diagnostic performance improvement. Our empirical case study shows that the static algorithm Lion considerably improves SFL's performance, compared to random placement, at a time complexity of $O(N \cdot M^3)$. Where Lion cannot exploit test execution results, the dynamic counterpart Tiger benefits from its integration in the diagnostic loop. Even with its low-cost heuristic, Tiger is capable of approaching optimum placement, roughly within a factor close to 3 in terms of monitors required. As this paper is only a first step into this new direction in SFL, future work includes further empirical study into the precise characteristics of both simple algorithms, as well as their improvements.

References

[Abreu and van Gemund, 2010] R. Abreu and A.J.C. van Gemund. Diagnosing intermittent faults using maximum likelihood estimation. *Artificial Intelligence*, 2010.

[Abreu *et al.*, 2007] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings TAIC-PART'07*, 2007.

[Abreu *et al.*, 2008] R. Abreu, A. Gonzalez, P. Zoeteweij, and A.J.C. van Gemund. Automatic software fault localization using generic program invariants. In *Proceedings ACM SAC'08*, 2008.

[Abreu *et al.*, 2009] R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 2009.

[Brodie *et al.*, 2003] M. Brodie, I. Rish, S. Ma, and N. Odintsova. Active probing strategies for problem diagnosis for distributed systems. *Proceedings IJCAI'03*, 2003.

[de Kleer and Williams, 1987] J. de Kleer and Brian C. Williams. *Diagnosing multiple faults*. 1987.

[de Kleer, 2007] J. de Kleer. Diagnosing multiple persistent and intermittent faults. In *Proceedings DX'07*, 2007.

[Feldman *et al.*, 2010] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, and A.J.C. van Gemund. Empirical evaluation of

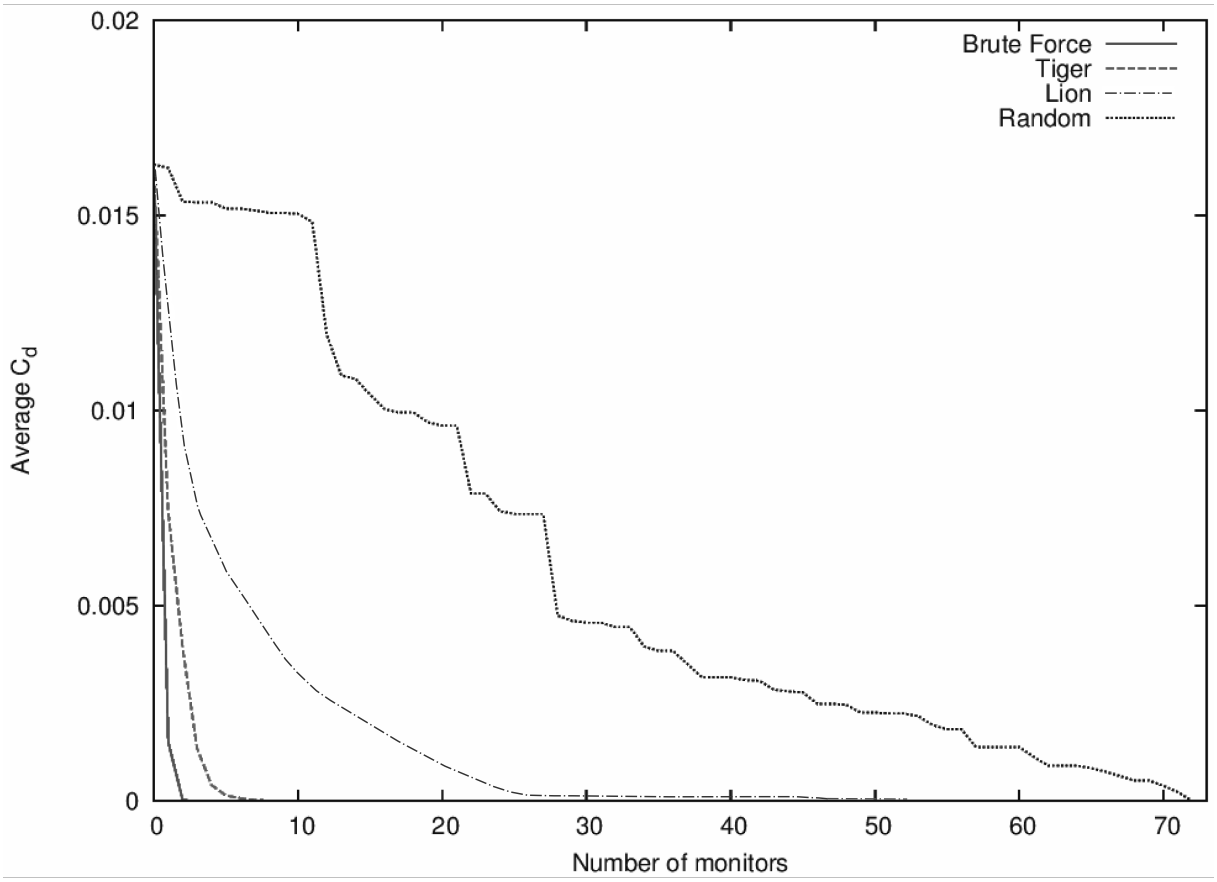


Figure 4: Performance Comparison

diagnostic algorithm performance using a generic framework. *International Journal of Prognostics and Health Management*, 2010.

[Feldman *et al.*, 2013] A. Feldman, H.Ṽincente de Castro, A.J.C. van Gemund, and G. Provan. Model-based diagnostic decision support system for satellites. In *Proceedings AeroConf'13*, 2013.

[Landi, 2012] C. Landi. Algoritmi per la riduzione dell'ambiguità di diagnosi topologiche, Università degli Studi di Brescia, Italy 2012. MSc thesis.

[Mathijssen, 2008] R.W.M. Mathijssen. Embedded Systems Institute, Personal Communication, 2008.

[Piel *et al.*, 2012] E. Piel, A. Gonzalez-Sanchez, H-G. Gross, A.J.C. van Gemund, and R. Abreu. Online spectrum-based fault localization for health monitoring and fault recovery of self-adaptive systems. In *Proceedings ICAS'12*, 2012.

[Pietersma and van Gemund, 2007] J. Pietersma and A.J.C. van Gemund. Benefits and costs of model-based fault diagnosis for semiconductor manufacturing equipment. In *Proceedings INCOSE'07*, 2007.

[Reeven, 2011] B.J. Reeven. Model-based diagnosis in industrial context, Delft University of Technology, The Netherlands 2011. MSc thesis.

[Rothermel *et al.*, 2001] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Softw. Eng.*, 2001.

[Schoemaker, 2008] E. Schoemaker. ASML Netherlands, Personal Communication, 2008.

[van der Laar *et al.*, 2013] P. van der Laar, J. Tretmans, and M. Borth. *Situation Awareness with Systems of Systems*. Springer, 2013.

[van Gemund *et al.*, 2011] A.J.C. van Gemund, S. Gupta, and R. Abreu. The ANTARES approach to automatic systems diagnosis. In *Proceedings DX'11*, 2011.

[Voas, 1996] J. Voas. Software testability measurement for intelligent assertion placement. *Software Quality Journal*, 1996.

[Wang *et al.*, 2009] X. Wang, S. Cheung, W. Chan, and Z. Zhang. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In *Proceedings ICSE'09*, 2009.

[Zoeteweij *et al.*, 2007] P. Zoeteweij, R. Abreu, R. Golsteijn, and A.J.C. van Gemund. Diagnosis of embedded software using program spectra. In *Proceedings ECBS'07*, 2007.

[Zoeteweij *et al.*, 2008] P. Zoeteweij, J. Pietersma, R. Abreu, A. Feldman, and A.J.C. van Gemund. Automated fault diagnosis in embedded systems. In *Proceedings SSIRI'08*, 2008.

A Kernel Density Estimate-Based Approach to Component Goodness Modeling

[†‡]**Nuno Cardoso** and [†‡]**Rui Abreu**
[†]Department of Informatics Engineering
Faculty of Engineering of University of Porto
Porto, Portugal
[‡]HASLab / INESC Tec
Campus de Gualtar
Braga, Portugal
nunopcardoso@gmail.com, rui@computer.org

Abstract

Intermittent fault localization approaches account for the fact that faulty components may fail intermittently by considering a parameter (known as goodness) that quantifies the probability that faulty components may still exhibit correct behavior. Current, state-of-the-art approaches (1) assume that this goodness probability is context independent and (2) do not provide means for integrating past diagnosis experience in the diagnostic mechanism. In this paper, we present a novel approach, coined Non-linear Feedback-based Goodness Estimate (NFGE), that uses kernel density estimations (KDE) to address such limitations. We evaluated the approach with both synthetic and real data, yielding lower estimation errors, thus increasing the diagnosis performance.

1 Introduction

Previous approaches to (automatic) fault localization often assumed faults to be persistent [3; 8; 12]. This is a fair assumption in scenarios where the understanding of the system and its model are very precise as, in theory, almost all systems are deterministic[1].

In practice, due to the inherent complexity of data modeling and analysis, most models are unable to distinguish all possible states the system can be in and, as a consequence, most faults appear to exhibit intermittent behavior (i.e., non-determinism). Concretely, an *intermittent fault* is a fault that, for equal system observations is not consistently triggered. This apparent intermittency was shown to greatly hamper the diagnosis power of persistent fault models [6].

To circumvent such limitation, the intermittent fault model was proposed in [13] and later applied in the scope of automatic software debugging in [6]. A key concept to the intermittency framework is the concept of *goodness*.

Definition 1 (Goodness)**.** *The goodness of a (faulty) component, g_j, is the probability that a component j, under a set of observable system variables, exhibits nominal behavior.*

Intermittent fault modeling was proved to have a better diagnostic accuracy than persistent fault in several domains [2; 14; 16]. In addition to the steps already required by the persistent fault framework (defined in Section 2), and due to the fact that these values are typically not available, the intermittency model raises the challenge of accurately estimating the goodness probability, g_j.

The great limitations of existent approaches relate to the fact that (1) g_j is estimated as being constant and independent from the system state and (2) such approaches do not provide any mechanism for improving the diagnostic performance given the outcome of previous diagnostics. In this paper we propose an approach that relies on a statistic non-parametric technique called Kernel Density Estimate (KDE) aimed at overcoming such limitations.

Results on synthetic data showed that our approach provides better goodness estimates than its constant counterparts and an overall low estimation error. Results on a real application showed that the accuracy of g_j plays a crucial role in the accuracy of the diagnosis system. The conducted real application tests proved that our approach was not only able to use the system's state to overcome the presented limitations of existent approaches but also that the accuracy of g_j was higher than for existent approaches.

This paper has been accepted at the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13) and makes the following contributions:

- We provide an overview of the state of the art approach to automatic software fault localization.
- We present a new approach for modeling g_j as a non-linear function using a set of feedback observations. The approach is called Non-linear Feedback-based Goodness Estimate (NFGE).
- We provide an empirical evaluation of our approach using both synthetic and real data.

The remainder of this paper is organized as follows. In Section 2 we introduce the concepts and definitions used throughout this paper as well as an overview of the current fault localization approach. In Section 3 we motivate and present our approach. In Section 4 we test our modeling approach with synthetic data and we evaluate the effects in the diagnosis of a real application with injected faults. Finally, in Section 5 we draw some conclusions about the paper.

2 Preliminaries

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

Definition 2 (Diagnosing System)**.** *A diagnostic system DS is defined as the triple $DS = (SD, COMPS, OBS)$, where:*

[1]At least outside the Quantum Mechanics world.

- *SD is a propositional theory describing the behavior of the system*
- *$COMPS = \{c_1, \ldots, c_M\}$ is a set of components in SD*
- *OBS is a set of observable variables in SD*

Definition 3 (h-literal)**.** *Under an observation term obs over variables in OBS, a component is considered healthy if it performed nominally and unhealthy otherwise. An h-literal, h_j for $c_j \in COMPS$, denotes the component's health.*

Definition 4 (h-clause)**.** *An h-clause is a disjunction of h-literals such that no pair of h-literals is complimentary.*

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

$$(\bigwedge_{m \in S_N} \neg h_m) \wedge (\bigwedge_{m \in S_P} h_m) \tag{1}$$

A diagnosis candidate for DS, given an observation term obs over variables in OBS, is $d(S_N, S_P)$ such that

$$SD \wedge obs \wedge d(S_N, S_P) \not\models \bot \tag{2}$$

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 6 (Diagnosis Report)**.** *A diagnosis $D = (d_1, \ldots, d_k, \ldots, d_K)$ is an ordered set of K diagnosis candidates, for which*

$$\forall d_k \in D : SD \wedge obs \wedge d_k \not\models \bot \tag{3}$$

The calculation of a diagnosis report can be broadly divided in two sub-problem: diagnosis candidate generation and ranking.

The candidate generation problem is normally solved by using search algorithms [1; 15; 10] that produce candidates that (1) are consistent with the observations and (2) heuristically, have a higher chance of being correct.

In the remainder of this section we describe the relevant aspects of the Bayesian approach to address ranking problem. We assume that a set of observations have been collected, either by static modeling or by dynamic profiling, both known as a *spectra* [11].

The far most commonly used type of spectra is called hit spectra [11]. Existent approaches [5; 4; 6; 14] use the hit spectra abstraction as input to the Bayesian probability update process.

Definition 7 (Hit Spectra)**.** *The hit spectra is a particular type of spectra that encodes involvement of each $c_j \in COMPS$ in transaction i in terms of hit/not hit. Formally, let N denote the total number of transactions and M denote the cardinality of $COMPS$. Let A denote the $N \times M$* activity matrix *of the system, defined as*

$$A_{ij} = \begin{cases} 1, & \textit{if component j was involved in transaction } i \\ 0, & \textit{otherwise} \end{cases} \tag{4}$$

Let e denote the error vector*, defined as*

$$e_i = \begin{cases} 1, & \textit{if transaction i failed} \\ 0, & \textit{otherwise} \end{cases} \tag{5}$$

In the scope of software diagnosis, for instance, a transaction is a sequence of component activity (e.g., statements, functions, services, etc.) that results in a particular return value.

T	LB	DB_1	HD_1	DB_2	HD_2	Error
1	1	1	1	0	0	1
2	1	0	0	1	1	1

Figure 1: Hit spectra example

Let p_j[2] denote the prior probability that a component c_j is at fault. Assuming components **fail independently**, the prior probability for a particular candidate $d \in D$ is given by

$$\Pr(d) = \prod_{j \in d} p_j \cdot \prod_{j \in COMPS \setminus d} (1 - p_j) \tag{6}$$

For a set of observations, obs[3], the posterior probabilities are calculated according to Bayes rule as

$$\Pr(d \mid obs) = \Pr(d) \cdot \prod_{obs_i \in obs} \frac{\Pr(obs_i \mid d)}{\Pr(obs_i)} \tag{7}$$

The denominator $\Pr(obs_i)$ is a normalizing term that is identical for all $d \in D$ and thus needs not to be computed directly. $\Pr(obs_i \mid d)$ is defined as

$$\Pr(obs_i \mid d) = \begin{cases} 0, & \text{if } obs_i, e_i, \text{ and } d \text{ are not consistent} \\ \varepsilon, & \text{otherwise} \end{cases} \tag{8}$$

ε [5; 14] can be defined as

$$\varepsilon = \begin{cases} 1 - \prod_{j \in (d \cap obs_i)} g_j & \text{if } e_i = 1 \\ \prod_{j \in (d \cap obs_i)} g_j & \text{otherwise} \end{cases} \tag{9}$$

In [6], the authors address this problem by maximizing $\Pr(obs \mid d)$ (Maximum Likelihood Estimation (MLE) for naive Bayes classifier) under parameters $\{g_j \mid j \in d\}$ for the above epsilon policy.

3 Approach

To motivate our approach and illustrate the limitations of existent approaches, consider an instance of the *Znn.com* case study [9], composed of a load balancer (LB) and a set of databases ($DB_1, \ldots, DB_x$) with respective hard drives ($HD_1, \ldots, HD_x$). The workload of such system consists in delivering static content to a set of clients. In this scenario each transfer represents a transaction and the success/failure of each transaction could be verified by computing a hash over the delivered content and checking it against a previously calculated hash.

The first limitation of existent approaches is related with the high level of abstraction enforced by the usage of hit spectra as it does not provide any information about the state of the system during each component's execution. Additionally, it abstracts the number of times each component was used and, consequently, the sequence in which they were used in each transaction. As a consequence, hit spectra-based approaches have difficulties in distinguishing pairs of components for which the activity is similar (i.e., columns present low entropy), such as for instance the databases and respective hard drives in Fig. 1. The occurrence of low entropy spectra is specially incident in environments where the

[2]The value of p_j is application dependent. In the context of development-time fault localization it is often approximated as $p_j = 1/1000$, i.e., 1 fault for each 1000 lines of code [7].

[3]Each row A_{i*} of the activity matrix encodes (in a binary form) the set $obs_i \in obs$.

nature of transactions is not easily controllable (e.g., runtime environments).

The second limitation of existent approaches relates to the fact that g_j is estimated as being constant with respect to a set of observations. Consider that the effective (normally unknown) goodness for the hard drives gradually decreases over time, as depicted in Fig. 2c. Due to the fact that the slope of the curve is low and the time is monotonic, g_j can, most of the times, be successfully approximated by a constant with small errors. However, if the observations spanned over a long period of time or the slope of the goodness curve was higher, a constant goodness function would fail to accurately model the actual goodness of the hard drive, entailing large average errors. Given the multiplicative nature of the goodness value usage, even small errors can have a serious impact in the diagnosis report ranking.

Finally, hit spectra-based approaches have limited possibilities of incorporating existent knowledge about the system in the diagnosis. As the state of the system is completely abstracted in the hit spectra, it would be impossible to distinguish a new hard drive from an old hard drive. As a consequence, even if the goodness curve was available for all components in the system, the algorithm would not be able to use it. Furthermore, in the event of all hard drives being failing (e.g., Fig. 1), existent approaches would assign a higher likelihood to the candidate containing only the load balancer as it has a lower cardinality than the correct candidate.

In the remainder of this section we present our approach, called Non-linear Feedback-based Goodness Estimate (NFGE), aimed at modelling the components' goodness as a non-linear function of a set of observable variables, $OBS_j \subseteq OBS$, referred to as $g_j(st)$. Additionally we present an improved version of ε that is used to plug $g_j(st)$ into the Bayesian framework to compute the posterior candidate probabilities $\Pr(d \mid obs)$ given a set of state spectra observations.

Definition 8 (State spectra)**.** *State spectra is a particular type of spectra that encodes the value of variables OBS_j for each execution of $c_j \in COMPS$. Formally, let N denote the total number of processes and M denote the cardinality of $COMPS$. Let A denote the $N \times M$* activity matrix *of the system, defined as*

$$A_{ij} = (st_1, ..., st_k, ..., st_K) \tag{10}$$

Each element of A_{ij}, st_k, encodes the value of variables OBS_j for the kth activation of component j in process i. Let e follow the definition presented in Def. 7.

Our approach can be divided into two independent stages: the modeling and diagnosis stages.

Modeling $g_j(st)$

Let an abstract data type, henceforward referred to as feedback spectra, be the data interface to our modeling approach.

Definition 9 (Feedback Spectra)**.** *The feedback spectra is a particular type of spectra that encodes the result of a set of diagnoses. Concretely, let M denote the cardinality of $COMPS$. FB_{ej} consists of a $2 \times M$ matrix defined as*

$$FB_{ej} = \{st_1, ..., st_k, ..., st_K\} \tag{11}$$

FB_{0j} and FB_{1j} are the pass and fail feedback observations sets respectively, for component c_j. Each element of FB_{ej} encodes the value of variables OBS_j.

In the following we assume that a mechanism for collecting feedback spectra exists. Additionally, we assume an atomic step of modeling that precedes all diagnosis.

Our approach to model $g_j(st)$ consists of a probabilistic model that is derived from the feedback spectra. Concretely, we estimate the pass and fail density functions, parametrized over variables in OBS_j[4], from which we trivially calculate $g_j(st)$.

The estimation of the pass/fail densities consist of a Kernel Density Estimate (KDE), $\hat{f}(st)$, and is described as

$$\hat{f}(st) = \frac{1}{bw} \sum_{c \in C} K\left(\frac{st - c}{bw}\right) \tag{12}$$

where C is a set of instantiations of OBS_j, $bw > 0$ is the bandwidth, a smoothing parameter, and $K(\cdot)$ is a kernel function[5]. A key aspect of KDE is the selection of the bandwidth parameter bw. In our approach, we estimate bw by using the Silverman's "rule of thumb" [17] defined as

$$0.9 \times min\left(\sigma, \frac{R}{1.34}\right) \times |C|^{(-0.2)} \tag{13}$$

where R is the interquartile range of sample C. Regarding $K(\cdot)$, even though several options exist, in our approach, we use the Gaussian kernel. Additionally, and without loss of generality, we will assume OBS_j contains only one variable.

As an example, consider the modeling process of $g_j(st)$ for a component c_j (e.g., the hard drives from the previous example) given 5 pass and 3 failed feedback observations with values $FB_{0j} = \{5, 7, 15, 20, 40\}$ and $FB_{1j} = \{40, 44, 60\}$, respectively.

The first step in modeling $g_j(st)$ is the estimation of the density function for the nominal executions depicted in Fig. 2a, formally defined in Eq. 12, with parameters $C = \{5, 7, 15, 20, 40\}$ and $bw = 6.328$. Note that for each value in the horizontal axis, the KDE value corresponds to the summation of all underlying kernels at the same point. From Eq. 12 we can see that C determines each kernel's offset and bw the dispersion of the density. In particular, when using the Gaussian kernel, C_i corresponds to its mean and bw to its standard deviation.

Fig. 2b provides a visual intuition on the effect of the parameter bw in the estimate. A sensible selection of bw is crucial in order to yield good results as using a small bw value will reflect sampling artifacts whereas a large bw value will smooth some behavioral trends.

The second and final step is the derivation of $g_j(st)$ from the pass/fail KDEs. We will assume that the previous step was repeated for the fail executions yielding the densities depicted in Fig. 2c. $g_j(st)$, depicted in black, is defined as

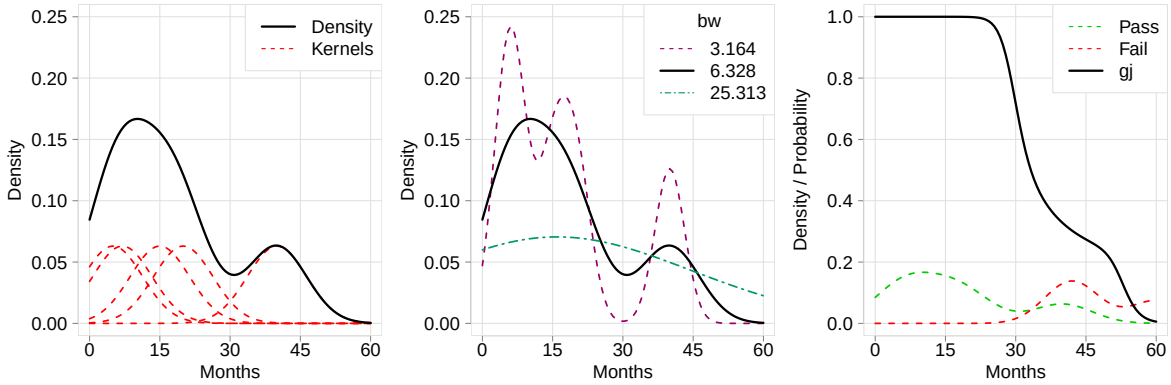
$$g_j(st) = \frac{\hat{f}_{\text{pass}}(st)}{\hat{f}_{\text{pass}}(st) + \hat{f}_{\text{fail}}(st)} \tag{14}$$

Ranking using $g_j(st)$

In order apply $g_j(st)$ to the Bayesian framework we define a new policy for ε. For a given process i and a set of state spectra observations, the epsilon policy is given by

[4] Currently, the variables are arbitrary and must be manually selected on a per-component basis.

[5] A kernel is a symmetric but not necessarily positive function that integrates to one.



(a) Density estimation and underlying kernels

(b) Impact of bw value

(c) Goodness vs. pass/fail KDEs

Figure 2: KDE visual intuition

$$\varepsilon = \begin{cases} 1 - \prod_{j \in d \wedge st_k \in A_{ij}} g_j(st_k) & \text{if } e_i = 1 \\ \prod_{j \in d \wedge st_k \in A_{ij}} g_j(st_k) & \text{otherwise} \end{cases} \quad (15)$$

In contrast to existent ε policies, our policy does not only take into account the state of c_j but also the number of times it was executed.

Complexity analysis

The time complexity of calculating the bandwidth parameter, bw, for a set of F feedback observations is $O(F \cdot log(F))$ (due to the calculation of the interquartile range). Consequently, the overall complexity of the modelling phase, for M components, would be $O(2 \cdot M \cdot F \cdot log(F))$. In practice, as the number of feedback observations needed to build an accurate model is small (< 50), we observe a maximum complexity of $O(2 \cdot f \cdot log(f) \cdot M)$, where f is a cutoff constant for the number of feedback observations.

Regarding the diagnosis phase, we can unfold it in simpler steps. For each component (M), transaction (T) and observation (S) we calculate the associated goodness ($O(M \cdot T \cdot S)$). The goodness estimation consists in calculating the result of Eq. 12 ($O(F) \approx O(f)$). Globally, the time complexity of this step is $O(f \cdot M \cdot T \cdot S)$.

4 Results

To assess the performance of our approach we conducted two studies. The first study is aimed at evaluating the prediction error of the modelling approach. At this stage, we use synthetic goodness models in order to be able to test a wider set of goodness patterns. After establishing the prediction error of our approach, the second study aims at exploring, in a real application, the cases where existent approaches tend to fail.

Prediction Error Study

To assess the prediction error of our approach we generated a set of 20000 random synthetic goodness models (M). With the purpose of having different learning and observation generation processes, we modified Eq. 12 such that each underlying kernel has an individual bw_i. Formally, the synthetic goodness models have the underlying pass and fail distributions defined as

$$\hat{f}(st) = \sum_{i=1}^{|C|} \frac{K(\frac{st - C_i}{bw_i})}{bw_i} \quad (16)$$

Additionally, in our synthetic data setup, two types of models can be distinguished. The first set of models use the Gaussian kernel as their building block. This kind of models are intended to mimic the behavior of components that exhibit smooth transitions between any two points in the feature space (i.e., the domain of the observable variables). This can be the case of component ageing in which the goodness normally decreases gradually with time (e.g., Fig. 2c).

The second type of models use the Box kernel, i.e., a rectangular-shaped kernel centered in C_i with bw_i width and $\frac{1}{bw_i}$ height. This set of models is intended to emulate components that exhibit abrupt transitions in their goodness functions. Also, as the original kernel differs from the learning kernel, the process of generation and learning becomes substantially different and enables drawing more meaningful conclusions.

Finally, the models generated range from simple patterns such as for instance the one depicted in Fig. 2c to more complex patterns with up to 20 supporting kernels for both the pass and fail densities.

To generate the feedback spectra, for each model we randomly selected a set of 200 values, F, in its feature space. For each of them we determine whether the component performed nominally in a Bernoulli trial process parameterized with $M_l(F_k)$. For each $M_l \in M$, we trained estimators with gradually increasing amounts of feedback spectra.

For each trained model, the prediction error is obtained by comparing the predicted goodness and the original goodness for 1000 evenly distributed samples of the feature space. The results are summarized by means of 4 metrics, as depicted in Fig. 3: absolute mean error, standard deviation, positive mean error and negative mean error. The first two plots depict the average results for NFGE in both the Gaussian and Box cases, respectively. The last plot depicts the average results for a constant estimator defined as

$$g_j(st) = \frac{|FB_{0j}|}{|FB_{0j}| + |FB_{1j}|} \quad (17)$$

As discussed, current approaches are not able to incorporate feedback information, rendering a direct comparison with NFGE impossible. Despite, we provide the results for the

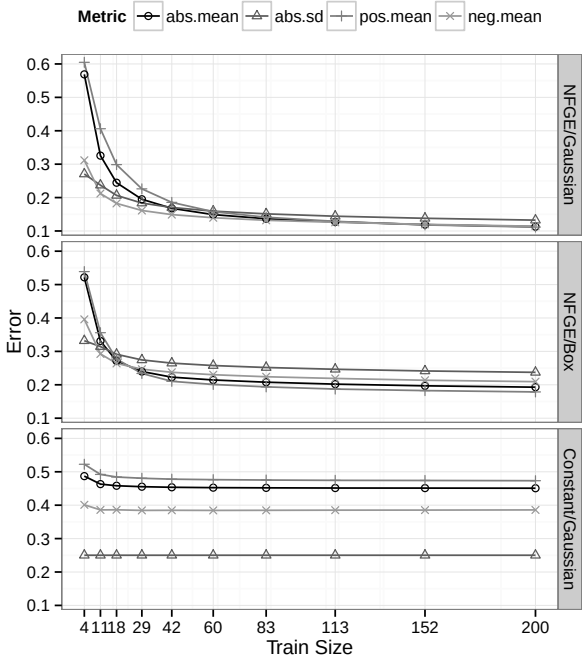


Figure 3: Prediction errors

constant estimator with the goal of establishing a ground of comparison to the hypothetical performance of existent algorithms if they were able to incorporate such information.

From the analysis of the results, we can see a clear improvement introduced by NFGE over the baseline results in all the observed metrics. As expected, the constant estimation presents a large average error of 44%. Additionally, the constant estimator does not scale with the size of the available feedback data.

In contrast, NFGE, in its best case scenario, was able to perform at 10% of average error. Globally, NFGE presented an average of 21% and 26% of prediction error for the smooth and non-smooth cases, respectively. The reason for the 5% difference relates to the fact that the gaussian-based KDE is not able, with a limited set of observations, to tightly fit the original box-based model.

Additionally a fairly quick convergence is observed: with 42 observations 90% of the maximum performance is obtained for both the Gaussian model/Gaussian estimator case and the Box model/Gaussian estimator case.

Finally, the results showed that when the amount of available data is small (< 11 observations), the constant estimate, on average, outperforms NFGE.

Diagnosis Study

In this section we evaluate the diagnosis performance of our approach in scenarios where existent approaches tend to fail. This study is mainly intended to serve as a proof of concept.

The selected application for this study was a simple http server, webfs[6], which was instrumented to generate state spectra. Additionally, we injected code to emulate the behavior of ageing faults (e.g., memory leaks, hard drive wear, etc.). The injected faults are parametrizable over 3 variables: min, max, and $total$. For each execution of each injected fault a counter is incremented with a random value greater than min and lesser than max. Whenever a counter reaches the value of its associated $total$ variable, the fault is triggered and the transaction fails. The counters may either be shared among a set of faults or, unless stated otherwise, independent.

To collect the feedback spectra required for generating the goodness models, we did a prior test run where the faults were targeted individually. For each injected fault execution, we recorded the number of previous invocations and process age, i.e., the time since the http server start. For each fault we created two univariate goodness models using 20 feedback observations: one as a function of the number of invocations and the other as a function of the process age. As we only created non-linear goodness estimates for the components with injected faults, for the remainder components we downgraded the state spectra into hit spectra and used the MLE approach (as presented in Section 2).

In a first scenario we only activated one fault. At this stage we tried to isolate the faulty component from a 5 element diagnosis report. The results showed that the MLE approach was only able to exonerate 1 out of 5 candidates. This was due to the fact that all transactions shared the same "bootstrap" sequence and the fault was injected in such sequence. As such components had the same activity pattern, i.e., equal columns in the hit spectra, and it remained impossible to distinguish them. In contrast, NFGE was able to clearly isolate the real faulty component. When comparing the candidates' probabilities for both approaches, NFGE calculated a probability of 99.9999% for the actual diagnosis whereas MLE calculated a probability $\approx 25\%$ for both the correct diagnosis and the 3 remaining components. This great difference in the likelihood magnitudes is in accordance with the results obtained in the synthetic experiments.

In a second scenario, we enabled two faulty components for which we had the corresponding goodness models. In this setup we generated a set of transactions that would eventually trigger one fault but not the other. Additionally, the two components were always activated in succession, generating low entropy hit spectra (as in Fig. 1). The goal of this test was to confirm that NFGE is both able to both indict and exonerate components depending on the execution context.

In this scenario, the MLE ranked equally the real error source and the component that should be exonerated. In contrast, NFGE was again able to both indict and exonerate components correctly.

Even though the previous setups produced positive results, it is important to note that the selection of the modeling variables is a crucial aspect. As previously stated we observed two distinct variables: the number of component executions and the process age. Even though both variables are able to model the goodness pattern of the injected faults, the degree to which each variable is able to cope with new scenarios may vary.

From this test setup it is easy to understand that even though correlated, the process age is not the cause of the fault activation: if a component is "old" but was never activated, the corresponding counter was never incremented. The process age variable indirectly encodes some aspects of the application workload. If a age based model was constructed for a specific workload, i.e., activations/second, its ability to produce positive results in a different workload

[6]Available at: *http://linux.bytesex.org/misc/webfs.html*

may be limited. On the other hand it is clear that the component activation count is much more independent from the application's workload.

In the previous setups we used independent counters. If on the other hand we had a global counter and still used independent activation counters, the same variable would also implicitly encode workload patterns. Furthermore, if the modeling workload is substantially different from the diagnosis workload it could happen that the components that caused the errors would be exonerated.

5 Conclusions

In this paper we presented a novel approach to fault localization that is able to absorb past diagnosis experience in order to improve its future diagnosis performance. Additionally, our approach, by using KDEs, is able to model the components' goodness as a non-linear function of a set of observable system variables.

A major improvement over hit spectra-based techniques is the possibility of using the value of the system variables to distinguish components that would otherwise be indistinguishable (due to high entropy spectra). Also, this enables the exoneration of faulty components in cases where the fault was not probably activated.

The conducted synthetic experiments produced promising results which were reproduced in the case study. However, from the real experiments we were able to clearly see some of the limitations of our approach.

Currently, the goodness modeling stage is still essentially manual. Work in devising tools for automatically determining the modeling variables of each component would have a deep impact on the usability of our technique. Also, it should be possible to extract information from previous diagnosis to automatically gather the feedback spectra.

Acknowledgements

We would like to thank Lígia Massena, André Silva, David Garlan, Bradley Schemerl, Paulo Casanova and Alexandre Perez for the useful feedback on previous versions of this paper. This material is based upon work supported by the National Science Foundation under Grant No. CNS 1116848, and by the scholarship number SFRH/BD/79368/2011 from Fundação para a Ciência e Tecnologia (FCT).

References

[1] Rui Abreu and Arjan van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Proceedings of the 8th Symposium on Abstraction, Reformulation, and Approximation*, SARA'09, 2009.

[2] Rui Abreu and Arjan van Gemund. Diagnosing multiple intermittent failures using maximum likelihood estimation. *Artificial Intelligence*, 174(18):1481–1497, 2010.

[3] Rui Abreu, Peter Zoeteweij, and Arjan van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques*, TAICPART'07, pages 89–98, 2007.

[4] Rui Abreu, Peter Zoeteweij, and Arjan van Gemund. A dynamic modeling approach to software multiple-fault localization. In *Proceedings of the 19th International Workshop on Principles of Diagnosis*, DX'08, pages 7–14, 2008.

[5] Rui Abreu, Peter Zoeteweij, and Arjan van Gemund. An observation-based model for fault localization. In *Proceedings of the 6th Workshop on Dynamic Analysis*, WODA'08, pages 64–70, 2008.

[6] Rui Abreu, Peter Zoeteweij, and Arjan van Gemund. A new bayesian approach to multiple intermittent fault diagnosis. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 653–658, 2009.

[7] John Carey, Neil Gross, Marcia Stepanek, and Otis Port. Software hell. In *Business Week*, pages 391–411, 1999.

[8] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, O Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN 2002, pages 595–604, 2002.

[9] Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. Evaluating the effectiveness of the rainbow self-adaptive system. In *Proceedings of Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'09, pages 132–141, 2009.

[10] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 911–918, 2008.

[11] Mary Jean Harrold, Gregg Rothermel, Rui Wu, and Liu Yi. An empirical investigation of program spectra. In *Proceedings of the 1998 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE'98, pages 83–90, 1998.

[12] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 273–282, 2005.

[13] Johan de Kleer. Getting the probabilities right for measurement selection. In *17th International Workshop on Principles of Diagnosis*, DX'06, pages 141–146, 2006.

[14] Johan de Kleer. Diagnosing multiple persistent and intermittent faults. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 733–738, 2009.

[15] Johan de Kleer and Brian C. Williams. Readings in model-based diagnosis. In *Readings in model-based diagnosis*, chapter Diagnosing multiple faults, pages 100–117. 1992.

[16] Lukas Kuhn, Bob Price, Minh Do, Juan Liu, Rong Zhou, Tim Schmidt, and Johan de Kleer. Pervasive diagnosis. *Transactions on Systems, Man Cybernetics - Part A*, 40(5):1562–1595, 2010.

[17] B. W. Silverman. *Density estimation for statistics and data analysis*. Chapman and Hall, 1986.

A Distributed Strategy for Deriving Minimal Hitting-Sets*

Xiangfu Zhao[1] and **Dantong Ouyang**[2]
[1]Department of Computer Science, Zhejiang Normal University, Jinhua, China
[2]School of Computer Science and Technology, Jilin University, Changchun, China
Email: xiangfuzhao@gmail.com, ouyd@jlu.edu.cn

Abstract

Deriving all minimal hitting-sets (MHSs) of a conflict set cluster is a classical problem of model-based diagnosis. A technique for distributed MHSs is proposed, based on an equivalence relation of elements. Then, a strategy for deriving all distributed MHSs is presented. If the set cluster is decomposed into a number of equivalence classes, parallel computation of MHSs for each distribution can be applied. An incremental, distributed approach is also proposed. Experimental results show that, compared with global methods, the efficiency for deriving all MHSs in a distributed way improves considerably.

1 Introduction

In model-based diagnosis [de Kleer and Williams, 1987; Reiter, 1987], given a system model from first principles, conflict sets of components are generated, by comparing the observation of the system and the prediction of the system model. Then, all minimal hitting-sets (MHSs), *i.e.*, candidate diagnoses, of those conflict sets, are derived. This is why deriving all MHSs is an important topic.

Several algorithms to derive all MHSs of conflict sets have been proposed, including HS-tree [Reiter, 1987], HS-DAG [Greiner *et al.*, 1989], HST-tree [Wotawa, 2001], S-TACCATO [Abreu and van Gemund, 2009], the approach based on a bipartite graph [de Kleer, 2011], BHS-tree [Lin and Jiang, 2003], Bool method [Lin and Jiang, 2003] along with relevant optimizations [Pill and Quaritsch, 2012], and our HSSE-tree [Zhao and Ouyang, 2006]. However, all of them are *global* ones, that is, *all* conflict sets at one time are to be considered for generating MHSs. Since it is NP-hard for computing all MHSs [Karp, 1972], with the complexity being exponential in the number of elements in all conflict sets, the computation is time-consuming when several conflict sets are given.

Alternatively, if a number of conflict sets are not related to others (for example, there is no common element between these conflict sets and others), then the MHSs of these conflicts will not be related to those ones. Therefore, to improve the efficiency of computing MHSs of all conflict sets, firstly we can find all the related conflict sets, and then compute all MHSs of the related ones with smaller size in parallel.

2 Preliminaries

In this section, we give some related definitions.

*This work was supported in part by NSFC under Grant Nos. 61003101 and 61272208; Zhejiang Provincial Natural Science Foundation under Grant No. Y1100191.

Definition 1. *Given a set cluster (set of sets) F, a set H is a hitting set for F, if the following conditions hold:*

(1) $H \subseteq \bigcup_{S \in F} S$;

(2) *For any set $S \in F$, $H \cap S \neq \emptyset$.*

If no proper subset of H is a hitting set for F, then H is called a minimal hitting set (MHS) for F.

We also use $MHS(F)$ to denote the set of all MHSs for F, that is, $MHS(F) = \{S \mid S \text{ is an MHS for } F\}$.

Example 1. Given a set cluster F = {{1, 2, 3}, {2, 4}, {3, 4}}, then we use any approach, such as HS-tree, HS-DAG, or Bool algorithm, *etc.*, to easily get $MHS(F)$ = {{1, 4}, {2, 3}, {2, 4}, {3, 4}}. ◇

In order to partition a set (cluster), we give a notion of equivalence class as follows.

Definition 2. *Given a set S and a binary equivalence relation R on S, an equivalence class of $x \in S$ is: $[x]_R$ = {$y \mid y \in S \wedge xRy$}, where xRy denotes x and y satisfy R.*

Example 2. Given a set S = {1, 2, 3 4} and a binary equivalence relation R = {$\langle 1, 1\rangle$, $\langle 1, 3\rangle$, $\langle 2, 2\rangle$, $\langle 2, 4\rangle$, $\langle 3, 1\rangle$, $\langle 3, 3\rangle$, $\langle 4, 2\rangle$, $\langle 4, 4\rangle$}, then $[1]_R = [3]_R = \{1, 3\}$, and $[2]_R = [4]_R = \{2, 4\}$. ◇

As for a set cluster, we give a new notion of join relation, by considering whether there are some elements *joining* two sets, as follows.

Definition 3. *Given a set cluster $F = \{S_1, S_2, \ldots, S_n\}$, a binary relation R is a join relation on F if, for any two sets $S_i, S_j \in F$ with $S_i R S_j$, such that: either $S_i \cap S_j \neq \emptyset$, or there exist $k_1, k_2, \ldots, k_l$ ($1 \leq l \leq n$), such that, $S_{k_1} \cap S_{k_2} \neq \emptyset$, $S_{k_2} \cap S_{k_3} \neq \emptyset$, ..., and $S_{k_{(l-1)}} \cap S_{k_l} \neq \emptyset$, where $k_1 = i$, and $k_l = j$.*

Proposition 1. *The join relation R on a nonempty set cluster $F = \{S_1, S_2, \ldots, S_n\}$ is an equivalence relation, and the equivalence class of S_i is: $\{S_j \mid S_j \in F \wedge S_i R S_j\}$.*

Example 3. Given a set cluster $F = \{\{1, 2, 3\}, \{2, 4\}, \{5, 6\}, \{6, 7\}\}$, all equivalence classes based on the join relation are: {{1, 2, 3}, {2, 4}}, and {{5, 6}, {6, 7}}. ◇

3 Deriving MHSs in a distributed way

In this section, a distributed strategy for deriving all MHSs is given, based on the join relation R.

3.1 Distributed MHS

In order to describe all MHSs of a set cluster, a notion of cross-production of set clusters is presented.

Algorithm 1 (Deriving all MHSs in distribution)

Input: A set cluster $F = \{S_1, S_2, \ldots, S_n\}$, and the join relation R on F;

Step 1: Call Function Distribute(F, R) to obtain equivalence classes: EC_1, EC_2, …, EC_k;
Step 2: For (i = 1; $i \leq k$; i ++)
MHS(EC_i) = **MHS_Gen_method**(EC_i);
EndFor

Output: Distributed MHSs as a k-tuple: (MHS(EC_1), MHS(EC_2), …, MHS(EC_k));

Definition 4. *Given two* nonempty *set clusters* $F_1 = \{S_{11}, S_{12}, \ldots, S_{1n}\}$ *and* $F_2 = \{S_{21}, S_{22}, \ldots, S_{2m}\}$, *the cross-production of* F_1 *and* F_2 *is denoted as:* $F_1 \times F_2 = \{S_{1i} \cup S_{2j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$.

Based on definitions of *Cross-production of set clusters* and *Minimal hitting set*, we have the following theorem.

Theorem 1. *Given a set cluster* F, *assume all equivalence classes of* F *based on the join relation* R *are* $EC_1, EC_2, \ldots, EC_k$ ($1 \leq k \leq |F|$, $|F|$ *is the length of* F). *Then,*

$$MHS(F) = \underset{1 \leq i \leq k}{\times} MHS(EC_i).$$

Example 4. Considering the set cluster F in Example 3, we get two equivalence classes of F: EC_1 = {{1, 2, 3}, {2, 4}}, and EC_2 = {{5, 6}, {6, 7}}. Thus, we easily get $MHS(EC_1)$ = {{2}, {1, 4}, {3, 4}}, and $MHS(EC_2)$ = {{6}, {5, 7}}.

According to Theorem 1, we have:

$$\begin{aligned} MHS(F) &= MHS(EC_1) \times MHS(EC_2) \\ &= \{\{2,6\},\{2,5,7\},\{1,4,6\},\{1, \\ &\quad 4,5,7\},\{3,4,6\},\{3,4,5,7\}\}. \end{aligned}$$

It's easy to verify $MHS(F)$ directly in a global way. ◇

From Theorem 1, we know that all MHSs of a set cluster is just the cross-production of the set cluster of MHSs for each equivalence class. In order to compactly represent all MHSs, we give a new notion of distributed MHS as follows.

Definition 5. *Given a set cluster* F, *and suppose all equivalence classes of* F *based on the join relation* R *are* $EC_1, EC_2, \ldots, EC_k$ ($1 \leq k \leq |F|$), *then the distributed MHS of* F *is denoted as a* k*-tuple:*

$$(MHS(EC_1), MHS(EC_2), \ldots, MHS(EC_k)).$$

Example 5. Considering the set cluster F in Example 4, after we generate $MHS(EC_1)$ and $MHS(EC_2)$ for the corresponding equivalence classes EC_1 and EC_2, according to Definition 5, all MHSs of F can be represented compactly as a pair $(MHS(EC_1), MHS(EC_2))$. ◇

3.2 Algorithms

In this subsection, **Algorithm 1** implements the distributed computation of all MHSs in detail.

Note: If **Step 2** in Algorithm 1 is implemented in parallel, it will be more efficient in practice. Function **MHS_Gen_method**(EC_i) is any one global method for generating all MHSs, given the conflict set cluster EC_i.

Function **Distribute**(F, R) is to decompose the set cluster F by the join relation R. The array EC_Arr_L denotes all the equivalence classes generated by R. In line 01, we initialize the first element of array EC_Arr_L $EC\_Arr\_L[1]$ to be $\{S_1\}$. The variable *NumofCurECs* represents the number of current equivalence classes, which is initialized to 1

Function Distribute(a set cluster F, a join relation R)

Input: a set cluster $F = \{S_1, S_2, \ldots, S_n\}$, a join relation R on F;

```
01: Init equivalence class array EC_Arr_L[1] = {S_1}
02:     and NumofCurECs = 1;
03: For(i = 2; i ≤ n; i + +)
04:   For(j = 1; j ≤ NumofCurECs; j + +)
05:     If(∃k ∈ N, s.t. S_k∈EC_Arr_L[j]∧S_iRS_k) Then
06:       EC_Arr_L[j] = EC_Arr_L[j] ∪ {S_i};
07:       Break;
08:     EndIf
09:   EndFor
10:   If(j > NumofCurECs) Then
11:     EC_Arr_L[++NumofCurECs] = {S_i};
12:   EndIf
13: EndFor
```

Output: The equivalence class array EC_Arr_L;

(line 02). From line 03, we start to check the rest sets in F one by one (S_i). For each equivalence class obtained by now (line 04), if we find some set in the equivalence class $EC\_Arr\_L[j]$ and S_i satisfy R, then S_i is added into $EC\_Arr\_L[j]$ (line 05 and 06). However, if S_i does not belong to any current equivalence class, we generate a new equivalence class for this set (line 10 and 11).

3.3 Complexity

Given a set cluster F, let n, num, and k denote the number of sets in F, the number of basic elements of sets in F, and the number of equivalence classes in F, respectively.

(1) The complexity of Function "Distribute" is mainly determined by the number of judgement, *i.e.*, judging whether S_i and some set in an equivalence class $EC\_Arr\_L[j]$ have the relation R. From the two "**For**" cycles (line 03 and 04), we know that the complexity is $O(n^2)$ in the worst cases.

Note: the complexity of checking S_iRS_j is $O(1)$ in line 05, since in practice we denote a set as an array of 0/1 digits (*e.g.*, if the max number of elements in a set is 8, and a set {1, 2, 3} is then denoted by an array: 00000111), and thus to check the join relation is equivalent to performing *one* binary bit operation "*And*" between the two arrays.

(2) For Algorithm 1, after calling function "Distribute" to generate all equivalence classes (**Step 1**), we produce all MHSs using a MHS-generating method (**Step 2**). The complexity of generating all MHSs is exponential in the number num_i of basic elements of sets in EC_i, *i.e.*, $O(2^{num_i})$, since it is an *NP-Complete* problem for generating all MHSs.

(3) By comparing the efficiency of computing distribution with the efficiency of generating all MHSs in theory, the complexity of the former ($O(n^2)$) is lower than the later ($O(2^{num_i})$) when $num_i \geq n$, even the time cost for computing distribution may be omitted (the implementation in Section 5 later give a clear evidence for this conclusion).

(4) The complexity of computing all MHSs of each class is $O(2^{(num/k)})$ in average. If k is very large, $O(2^{(num/k)})$ will be much lower than $O(2^{num})$ (the complexity of computing all MHSs in a global way). Therefore, the efficiency is improved considerably by computing all MHSs in a distributed way than in a global way, when there are a large number of equivalence classes in the set cluster.

4 Incrementally distributed solutions

In this section, an incremental strategy for updating distributed MHSs is presented.

4.1 Formalization and an example

Given a set cluster F, assume all equivalence classes of F based on R are $EC_1, EC_2, \ldots, EC_k$ $(1 \leq k \leq |F|)$. Then, the distributed MHSs of F is a k-tuple:

$$(MHS(EC_1), MHS(EC_2), \ldots, MHS(EC_k)).$$

Suppose a new set S_N is added into F, in order to keep the join relation R on F, F is updated as follows.

$$\begin{aligned} F \cup \{S_N\} &= EC_1 \uplus EC_2 \uplus \ldots EC_k \cup \{S_N\} \\ &= EC_{new} \uplus EC_{i(q+1)} \uplus EC_{i(q+2)} \uplus \ldots EC_{ik} \end{aligned}$$

where the symbol "$\uplus$" denotes the union of two disjoint sets (according to equivalence classes); and $EC_{new} = EC_{i1} \uplus EC_{i2} \uplus \ldots \uplus EC_{iq} \cup \{S_N\}$, where: for each EC_{ip} $(1 \leq p \leq q)$, there exists at least a set $S \in EC_{ip}$, such that: $S \cap S_N \neq \emptyset$.

In other words, EC_{new} is the only *new* equivalence class, composed with S_N and all the corresponding equivalence classes in F, each of which is related to S_N. Therefore, all equivalence classes of F, after S_N is inserted, are updated as:

$$EC_{new}, EC_{i(q+1)}, EC_{i(q+2)}, \ldots, EC_{ik}.$$

In order to generate all MHSs of F, after S_N is inserted, in a distributed way, we only need to generate all MHSs of the new equivalence class EC_{new}: $MHS(EC_{new})$.

A *global* method to incrementally derive all MHSs [Bailey and Stuckey, 2005], can be rephrased as follows.

$$\begin{aligned} & MHS(EC_{new}) \\ = \; & Min\left(\overset{q}{\underset{p=1}{\times}} MHS(EC_{ip}) \times \{\{e\} \mid e \in S_N\}\right) \quad (1) \end{aligned}$$

where function Min is used to make the set cluster minimal (by deleting all the proper supersets of any set in the cluster).

In other words, we first generate all MHSs of $EC_{i1}, \ldots, EC_{iq}$ by cross-production. Then, we extend each MHS with an element of S_N. Finally, we check them to delete all the proper supersets.

However, this method is somewhat complex, since it does not consider the distributed character of equivalence classes but in a global way (by cross-production). Thus, the efficiency is very low (we give comparisons between such a global approach with our distributed way in Section 5).

In the following, a novel method to efficiently generate all MHSs of EC_{new} is given, based on just the MHSs of each equivalence class EC_{ip} $(1 \leq p \leq q)$.

The basic idea of our method is simply explained as follows: Firstly, divide each $MHS(EC_{ip})$ into two disjoint parts (M_{p1} and M_{p2}), where each set of M_{p1} does not "hit" S_N, and each set of M_{p2} "hits" S_N. Then, each MHS of EC_{new} is one of the following two cases:

(1) An MHS is a union of q sets, each of which is from $MHS(EC_{ip})$ $(1 \leq p \leq q)$ respectively, and at least one of which is from M_{p2} ("hitting" S_N).

(2) An MHS is a union of q sets, each of which is from M_{p1} $(1 \leq p \leq q$; not "hitting" $S_N)$, and then extended with one element of S_N.

Formally, for each EC_{ip} $(1 \leq p \leq q)$, $MHS(EC_{ip}) = M_{p1} \uplus M_{p2}$, where $M_{p1} = \{S \mid S \in MHS(EC_{ip}), S \cap S_N = \emptyset\}$, and $M_{p2} = \{S \mid S \in MHS(EC_{ip}), S \cap S_N \neq \emptyset\}$. In order to introduce our approach more conveniently, some symbols are given: $S_N^{'} = \{\{e\} \mid e \in S_N\}$, representing the set cluster, each of which is composed just by a single element in S_N, and $SEC_{ip} = \bigcup EC_{ip}$ representing all basic elements in the sets of cluster EC_{ip}. Also let $S_{N_{ip}}^{'} = \{\{e\} \mid e \in SEC_{ip} \cap S_N\}$, denoting all the single element sets, and each element is the common one between all sets of EC_{ip} and S_N. Let $NewE = \{\{e\} \mid e \in S_N - \overset{q}{\underset{p=1}{\uplus}} SEC_{ip}\}$, representing all the single element sets, each of which is *completely* new one never appearing in F before. Then, we get: $S_N^{'} = \overset{q}{\underset{p=1}{\uplus}} S_{N_{ip}}^{'} \uplus NewE$.

We give the fundamentals of computation of MHSs for the new equivalence class EC_{new} as follows (note: $S_1 \times (S_2 \| S_3) = (S_1 \times S_2) \cup (S_1 \times S_3)$ in the following):

$$\begin{aligned} & MHS(EC_{new}) \\ = \; & M_{11} \times M_{21} \times \ldots \times M_{(q-1)1} \times M_{q2} \uplus \\ & \ldots \uplus \\ & M_{11} \times M_{21} \times \ldots \times M_{(i-1)2} \times (M_{i1} \| M_{i2}) \times \\ & (M_{(i+1)1} \| M_{(i+1)2}) \times \ldots \times (M_{q1} \| M_{q2}) \uplus \\ & \ldots \uplus \\ & M_{12} \times (M_{21} \| M_{22}) \times \ldots \times (M_{q1} \| M_{q2}) \uplus \\ & (M_{11} \times M_{21} \times \cdots \times M_{q1} \propto S_N^{'}) \\ = \; & \overset{q}{\underset{p=1}{\uplus}} \left(\overset{p-1}{\underset{j=1}{\times}} M_{j1} \times M_{p2} \times \overset{q}{\underset{j=p+1}{\times}} (M_{j1} \| M_{j2}) \right) \uplus \\ & \left(\overset{q}{\underset{p=1}{\times}} M_{p1} \propto S_N^{'} \right) \quad (2) \end{aligned}$$

Sub-expression $\overset{q}{\underset{p=1}{\uplus}} \left(\overset{p-1}{\underset{j=1}{\times}} M_{j1} \times M_{p2} \times \overset{q}{\underset{j=p+1}{\times}} (M_{j1} \| M_{j2}) \right)$ denotes the first case of MHSs of EC_{new}, each of which is from M_{p1} or M_{p2} $(1 \leq p \leq q)$, and at least one is from some M_{p2} ("*hitting*" S_N).

The later sub-expression $\left(\overset{q}{\underset{p=1}{\times}} M_{p1} \propto S_N^{'} \right)$ represents the second case of MHSs of EC_{new}, each MHS is extended by an element in S_N, is defined as follows (note: here the symbol "$\propto$" can be seen as "extended by").

$$\begin{aligned} & \overset{q}{\underset{p=1}{\times}} M_{p1} \propto S_N^{'} \\ = \; & \overset{q}{\underset{p=1}{\times}} M_{p1} \propto \left(\overset{q}{\underset{p=1}{\uplus}} S_{N_{ip}}^{'} \uplus NewE \right) \\ = \; & \left(\overset{q}{\underset{p=1}{\times}} M_{p1} \propto \overset{q}{\underset{p=1}{\uplus}} S_{N_{ip}}^{'} \right) \uplus \left(\overset{q}{\underset{p=1}{\times}} M_{p1} \propto NewE \right), \end{aligned}$$

where the sub-expression

$$\begin{aligned} & \overset{q}{\underset{p=1}{\times}} M_{p1} \propto \overset{q}{\underset{p=1}{\uplus}} S_{N_{ip}}^{'} \\ = \; & \overset{q}{\underset{p=1}{\uplus}} \left(\overset{p-1}{\underset{j=1}{\times}} M_{j1} \times \left(M_{p1} \nabla S_{N_{ip}}^{'} \right) \times \overset{q}{\underset{j=p+1}{\times}} M_{j1} \right), \end{aligned}$$

represents the distributed extension of M_{p1} with $S_{N_{ip}}^{'}$, *i.e.*, here we update every M_{p1} *only* with its corresponding $S_{N_{ip}}^{'}$ in parallel, where $M_{p1} \nabla S_{N_{ip}}^{'} = \{m_{p1} \uplus s_{n_{ip}}^{'} \mid m_{p1} \in M_{p1}, s_{n_{ip}}^{'} \in S_{N_{ip}}^{'}, \forall m_{p2} \in M_{p2} \rightarrow m_{p2} \not\subseteq (m_{p1} \uplus s_{n_{ip}}^{'})\}$, and the symbol "$\nabla$" is used to avoid generating any superset of some set in the corresponding M_{p2}.

The set cluster denoted by $\left(\overset{q}{\underset{p=1}{\times}} M_{p1} \propto NewE \right)$ denotes all the MHSs, each of which is composed by all M_{p1} and any one *completely* new element in S_N. Then we have:

$$\overset{q}{\underset{p=1}{\times}} M_{p1} \propto NewE = \overset{q}{\underset{p=1}{\times}} M_{p1} \times NewE.$$

Example 6. Given a set cluster F = {{1, 2}, {2, 3}, {4, 5, 6}, {5, 7}, {8, 9}}, according to the join relation R, F is divided as 3 equivalence classes: EC_1: {{1, 2}, {2, 3}}, EC_2: {{4, 5, 6}, {5, 7}}, and EC_3: {{8, 9}}. Then we get the corresponding MHS clusters: $MHS(EC_1)$: {{2}, {1, 3}}, $MHS(EC_2)$: {{5}, {4, 7}, {6, 7}}, and $MHS(EC_3)$: {{8}, {9}}. Thus, MHS(F) is denoted in a distributed way as a 3-tuple ($MHS(EC_1)$, $MHS(EC_2)$, $MHS(EC_3)$).

Now, suppose a new set S_N: {3, 5, 10} is added into F. Then the representation of new F and its corresponding distributed MHS clusters is updated as follows.

(1) Firstly, find all the equivalence classes of F related to S_N: EC_1 and EC_2. Then, EC_{new} is generated by combing the two classes with S_N: {{1, 2}, {2, 3}, {4, 5, 6}, {5, 7}, {3, 5, 10}}. Thus, the new F is divided to be two equivalence classes: EC_{new} and EC_3.

(2) For each $MHS(EC_{ip})(1 \leq p \leq 2)$ of EC_{new}, it is divided into two disjoint parts M_{p1} and M_{p2} as follows:

M_{11}: {{2}}, and M_{12}: {{1, 3}};

M_{21}: {{4, 7}, {6, 7}}, and M_{22}: {{5}};

(3) According to the definitions of following symbols, we get S'_N = {{3}, {5}, {10}}, S'_{N_1} = {{3}}, S'_{N_2} = {{5}}, $NewE$ = {{10}}.

(4) Now, we compute all the MHSs $MHS(EC_{new})$ of EC_{new} according to Equation (2), and finally we get:

$$\begin{aligned} & MHS(EC_{new}) \\ = \; & (M_{11} \times M_{22}) \uplus (M_{12} \times M_{21}) \uplus (M_{12} \times M_{22}) \\ & \uplus \left(\left(M_{11} \nabla S'_{N_1}\right) \times M_{21}\right) \uplus \left(M_{11} \times \left(M_{21} \nabla S'_{N_2}\right)\right) \\ & \uplus (M_{11} \times M_{21} \times NewE) \\ = \; & \{\{2,5\},\{1,3,4,7\},\{1,3,6,7\},\{1,3,5\},\{2,3, \\ & 4,7\},\{2,3,6,7\},\{2,4,7,10\},\{2,6,7,10\}\} \end{aligned}$$

(5) In the end, we get $MHS(F \uplus \{S_N\})$ in a distributed way to be a 2-tuple: ($MHS(EC_{new})$, $MHS(EC_3)$). ◇

4.2 Complexity

In order to better understand the complexity of our incremental method, we compare Equation (1) with Equation (2). In order to derive all MHSs, we mainly use only two kinds of operations: one is the union of two sets, and the other one is the minimization of a set cluster. Thus, we just compare the number of the set union and the number of minimization of a set cluster between the two methods.

For the global method represented by Equation (1), the number N_{UG} of set union is denoted as:

$$N_{UG} = \prod_{p=1}^{q} (|M_{p1}| + |M_{p2}|) * |S_N| \tag{3}$$

Whereas the number N_{UD} of set union in the distributed method represented by Equation (2) is depicted as:

$$\begin{aligned} N_{UD} \;=\; & \prod_{p=1}^{q} (|M_{p1}| + |M_{p2}|) \\ & + (|S_N| - 1) * \prod_{p=1}^{q} |M_{p1}| \end{aligned} \tag{4}$$

From Equation (3) and Equation (4), we have:

$$\begin{aligned} N_{UG} - N_{UD} \;=\; & \left(\prod_{p=1}^{q} (|M_{p1}| + |M_{p2}|) - \prod_{p=1}^{q} |M_{p1}|\right) \\ & * (|S_N| - 1) \end{aligned} \tag{5}$$

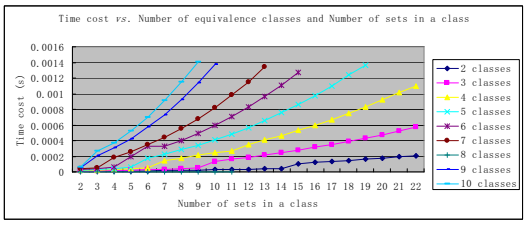


Figure 1: Time cost *vs.* Number of equivalence classes and Length of each equivalence class.

From Equation (5), a large number of set union operation will be reduced by the distributed method than the global method, if the length of the new set S_N is bigger than 1.

As to the number of minimization of a set cluster, that is, to find how many times of checking set containment. For the global method represented by Equation (1), the number N_{CG} of checking set containment in the worst cases is:

$$N_{CG} = N_{UG} * (N_{UG} - 1) \tag{6}$$

Whereas the number N_{CD} of checking set containment in a distributed way represented by Equation (2) is:

$$\begin{aligned} N_{CD} \;=\; & \sum_{p=1}^{q} \left(|M_{p1}| * |S'_{N_{ip}}| * |M_{p2}|\right) \\ \leq \; & |S_N| * \sum_{p=1}^{q} (|M_{p1}| * |M_{p2}|) \end{aligned} \tag{7}$$

Also, from Equations (3), (6), and (7), we know that a very large number of set containment is reduced by the distributed method, compared with the global method.

5 Implementation

The algorithms presented in this paper were implemented. The experiments are carried out on a PC Dell OptiPlex 780 with Intel(R) Core(TM)2 Quad CPU Q9400@2.66GHz (4 CPUs), 2046MB RAM, *Windows XP*, and *C++*.

5.1 Artificial examples

We suppose that the maximum number of components is 100, and they are marked by 0, 1, $\cdots$, 99. We also suppose that the number of all equivalence classes is from 2 to 10. For each set cluster Di_j, which is divided into i ($2 \leq i \leq 10$) equivalence classes, and there are $(j+1)$ sets with the same cardinality j ($2 \leq j < 100/i$) in each set, which are cyclic, and in the k^{th} ($1 \leq k \leq i$) class, all the $(j+1)$ sets are like: $\{(k-1)*(j+1), (k-1)*(j+1)+1, \ldots, (k-1)*(j+1)+j-1\}$, $\{(k-1)*(j+1)+1, (k-1)*(j+1)+2, \ldots, (k-1)*(j+1)+j\}$, $\{(k-1)*(j+1)+2, (k-1)*(j+1)+3, \ldots, (k-1)*(j+1)+j, (k-1)*(j+1)\}, \ldots, \{(k-1)*(j+1)+j, (k-1)*(j+1), \ldots, (k-1)*(j+1)+j-2\}$.

Efficiency of distributed decomposition

The efficiency of computing distribution is shown in Fig.1. From Fig.1, we can see that the efficiency is very high. In addition, on the one hand, the more sets in an equivalence class, the more time cost needed for decomposition; on the other hand, the bigger number of equivalence classes, the more time cost for decomposition, too.

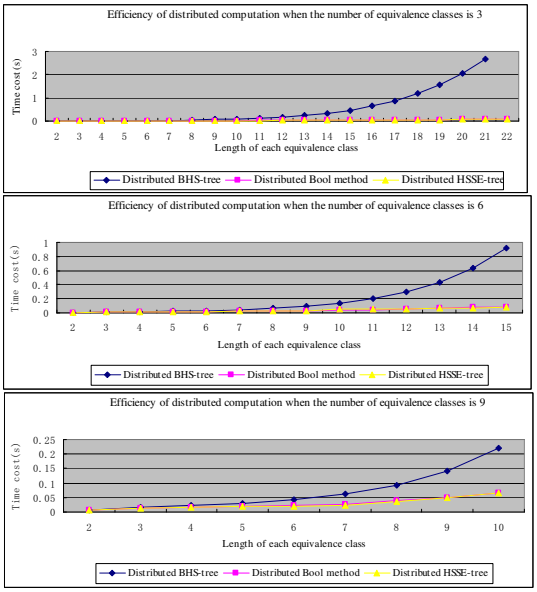


Figure 2: Distributed computation of MHSs by 3 methods.

Table 1: Time cost for Classification and Computation of MHSs with global or distributed methods (*s*) (data 1). Note: "-" denotes "out of RAM space".

Type	D9_2	D9_3	D9_4	D9_5	D9_6
Classification	**0.0001**	**0.0002**	**0.0003**	**0.0004**	**0.0006**
Global BHS-tree	42.896	> 600	> 600	> 600	> 600
Distributed BHS-tree	***0.0070***	***0.0152***	***0.0229***	***0.0306***	***0.0426***
Global Bool method	22.19	-	-	-	-
Distributed Bool method	***0.0062***	***0.0131***	***0.0177***	***0.0194***	***0.0216***
Global HSSE-tree	-	-	-	-	-
Distributed HSSE-tree	***0.0063***	***0.0128***	***0.0174***	***0.0193***	***0.0214***

Distributed computation of MHSs

Based on the equivalent decomposition of a set cluster, we generate all MHSs for each equivalence class with BHS-tree, Bool method, and HSSE-tree, respectively. Shown in Fig. 2 is the efficiency of the three methods in a distributed way, when the numbers of equivalence classes are selected as 3, 6, and 9. From Fig. 2, we know that the efficiency for computing MHSs in a distributed way is high. The time cost is also growing with the length of each equivalence class.

Distributed *vs*. Global computation

We compared the three methods (BHS-tree, Bool method, and HSSE-tree) both in global and in distributed way. Shown in Fig. 3 is the time cost for the selected number of equivalence classes 2 and 3, respectively. Based on Fig. 3, the computation of MHSs in a distributed way is more efficient than the corresponding global method. It becomes more and more evident when each equivalence class becomes larger and larger.

For the time cost of distribution of a set cluster and the computation of MHSs, from Table 1 and Table 2, we can see that the time cost of distribution (classification) is much smaller than the computation of MHSs with any method. Any time cost of distribution is less than *1%* of time cost of the corresponding global computation. Then, the time cost of distribution can be *ignored*, compared with the corresponding global computation of MHSs. Therefore, it is extremely *worthwhile* generating the equivalence classes before immediate computation of all MHSs in a global way.

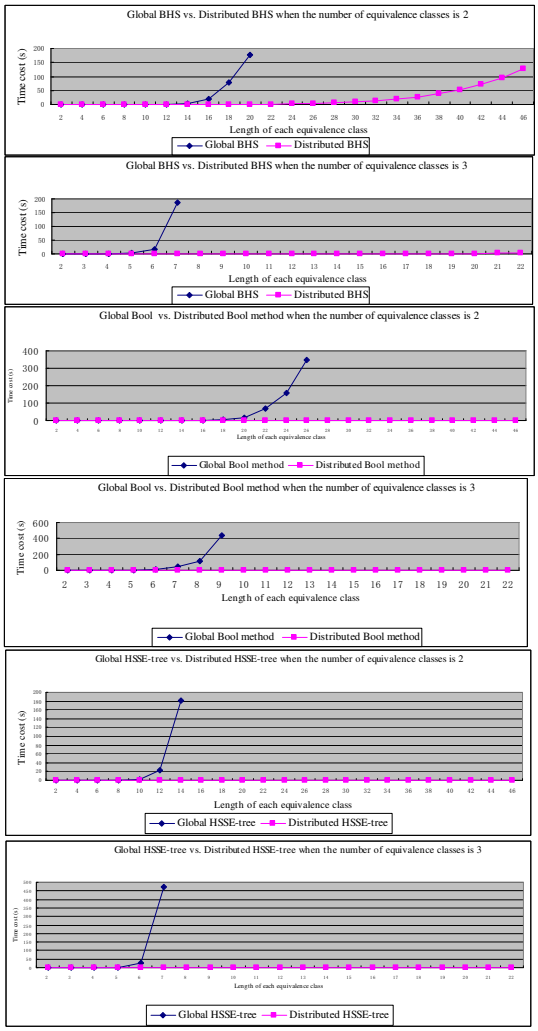


Figure 3: Comparisons of efficiency between the global way and the distributed way for each method.

Incremental update

Firstly, we compare the incremental computation of MHSs with the global (*i.e.*, non-incremental) computation of MHSs, both in a distributed way. Then, we compare the distributed update and the global update, both in an incremental way.

For the $(j+1)$ conflict sets like: $\{ (k-1)*(j+1), (k-1)*(j+1)+1, \ldots, (k-1)*(j+1)+j-1 \}$, $\{ (k-1)*(j+1)+1, (k-1)*(j+1)+2, \ldots, (k-1)*(j+1)+j \}$, $\{ (k-1)*(j+1)+2, (k-1)*(j+1)+3, \ldots, (k-1)*(j+1)+j, (k-1)*(j+1) \}, \ldots, \{ (k-1)*(j+1)+j, (k-1)*(j+1), \ldots, (k-1)*(j+1)+j-2 \}$ in the k^{th} class $(1 \leq k \leq i)$, we suppose that the new set S_N is the first set $\{ (k-1)*(j+1), (k-1)*(j+1)+1, \ldots, (k-1)*(j+1)+j-1 \}$ among the $(j+1)$ sets.

The time cost of Distributed BHS-tree, Distributed Bool method, Distributed HSSE-tree, and the *incremental* distributed method is shown in Fig. 4. From Fig. 4, we can see that incremental computation has the best efficiency, especially when the size of an equivalence class is larger and larger.

The time cost of the global incremental update and the distributed incremental update is shown in Fig. 5. From Fig. 5, we can see that distributed incremental update has the better efficiency than the global one, especially when

Table 2: Time cost for Classification and Computation of MHSs with global or distributed methods (s) (data 2). Note: "-" denotes "out of RAM space".

Type	D10_2	D10_3	D10_4	D10_5	D10_6
Classification	**0.0001**	**0.0003**	**0.0004**	**0.0005**	**0.0007**
Global BHS-tree	404.04	> 600	> 600	> 600	> 600
Distributed BHS-tree	***0.0080***	***0.0177***	***0.0238***	***0.0328***	***0.0460***
Global Bool method	217.28	-	-	-	-
Distributed Bool method	***0.0071***	***0.0151***	***0.0178***	***0.0206***	***0.0226***
Global HSSE-tree	-	-	-	-	-
Distributed HSSE-tree	***0.0072***	***0.0151***	***0.0183***	***0.0202***	***0.0221***

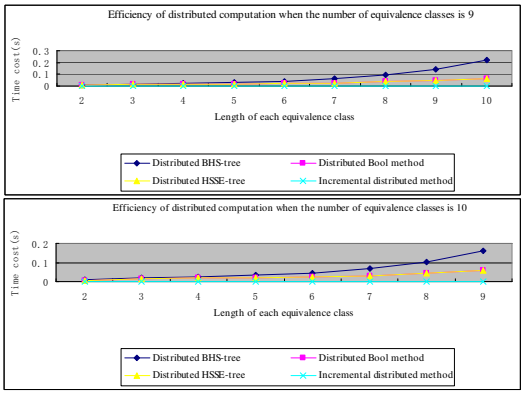


Figure 4: Comparisons of efficiency between the Incremental computation and the Non-Incremental computation.

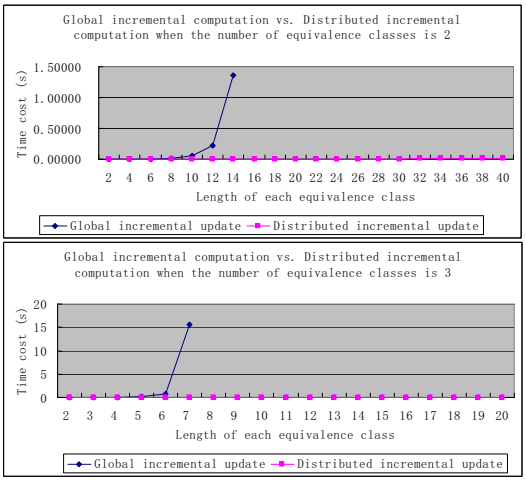


Figure 5: Comparisons of efficiency between the global way and the distributed way for incremental update.

the size of an equivalence class is larger and larger.

5.2 ISCAS-85 benchmark conflict sets

We also tested the conflict sets of benchmark circuits of ISCAS-85. We find most of conflict sets for circuits c5315, c880, c7522, c2670, and 74182 of ISCAS-85, are decomposed into many equivalence classes, and some of them are even divided into more than 20 classes. The ratios of decomposed conflict sets to all conflict sets are about 96%, 96%, 95%, 93%, and 76%, respectively for the five circuits. In general, the time cost is saved considerably, and some of them are about 99% time saved. There are a lot of conflict sets not be solved because of too much time cost (> 10 minutes) or short of RAM space by global methods, however, most of them are solved by the distributed methods.

6 Conclusion

A join relation is given, based on which, a set cluster can be divided into some equivalence classes. Then, all MHSs of the set cluster are derived independently on the equivalence classes in smaller size. Also, an incremental approach for updating MHSs is presented in a distributed way. Experimental results show that the approach in a distributed way is more efficient than in a global way, when the set cluster is decomposed into several equivalence classes.

Remark *Proofs of all Propositions and Theorems are available, yet omitted for space reasons.*

Acknowledgment

We owe a debt of gratitude to Dr. Rui Abreu and Dr. Alexander Feldman, for providing conflict sets of ISCAS-85 benchmarks; and to Prof. Gianfranco Lamperti for improvements of the paper.

References

[Abreu and van Gemund, 2009] R. Abreu and A. van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Proceedings of the 8th Symposium on Abstraction, Reformulation, and Approximation (SARA-09)*, pages 2–9, Murnau, Germany, 2009.

[Bailey and Stuckey, 2005] J. Bailey and P.J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of 7th international conference on Practical Aspects of Declarative Languages Pages (PADL-05)*, pages 174–186, Long Beach, CA, USA, 2005.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[de Kleer, 2011] J. de Kleer. Hitting set algorithms for model-based diagnosis. In *Proceedings of the 22ed International Workshop on Principles of Diagnosis (DX-11)*, pages 100–105, Murnau, Germany, 2011.

[Greiner *et al.*, 1989] R. Greiner, B. Smith, and R. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41:79–88, 1989.

[Karp, 1972] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, New York, 1972.

[Lin and Jiang, 2003] L. Lin and Y. Jiang. The computation of hitting sets: review and new algorithms. *Information Processing Letters*, 86:177–184, 2003.

[Pill and Quaritsch, 2012] I. Pill and T. Quaritsch. Optimizations for the boolean approach to computing minimal hitting sets. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)*, pages 648–653, Montpellier, France, 2012.

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

[Wotawa, 2001] F. Wotawa. A variant of reiter's hitting-set algorithm. *Information Processing Letters*, 79:45–51, 2001.

[Zhao and Ouyang, 2006] X. Zhao and D. Ouyang. A method of combining se-tree to compute all minimal hitting sets. *Progress in Natural Science*, 16:169–174, 2006.

Comparing Diagnostic Performance of Ochiai and Relief in Service-oriented Systems

Cuiting Chen[1], Brian Omoro[2], Hans-Gerhard Gross[1] and Andy Zaidman[1]
Delft University of Technology, the Netherlands
e-mail[1]: {cuiting.chen; h.g.gross; a.e.zaidman}@tudelft.nl e-mail[2]: b.o.omoro@student.tudelft.nl

Abstract

The dynamic nature of service-oriented systems necessitates efficient online techniques to diagnose and resolve operation time failures. Spectrum-based fault localization (SFL) is a diagnosis technique able to identify misbehaving services during runtime. Similarity coefficients (SC) such as Ochiai represent an essential element for the performance of SFL. Recently, also feature selection (FS) techniques such as Relief were proposed as more effective replacement for the SC. In this paper, we compare the diagnostic accuracy of Ochiai and Relief in a service-oriented context. Our results from system simulations and a case study indicate that the diagnostic capabilities of both approaches differ with respect to the system set-up and the number of faults.

1 Introduction

The highly dynamic nature of service-oriented systems facilitates the construction of dynamic, adaptable, and evolvable systems [Canfora and Di Penta, 2009]. However, such a dynamic nature also brings challenges in quality assurance. For example, operation-time reconfiguration may also lead to service failure, that should be detected and recovered dynamically. Failure recovery, on the other hand, demands adequate monitoring techniques to be built into the service-based system plus a diagnosis approach, in order to be able to identify upcoming service reconfiguration issues, and to pinpoint the misbehaving service to be exchanged.

Spectrum-based fault localization (SFL) in combination with adequate monitoring strategies has been shown to perform well in pinpointing faulty service operations [Chen *et al.*, 2012]. SFL works by inferring a *diagnosis* from symptoms: a ranking of potentially faulty services in a system, with the most likely faulty one ranked top. The *symptoms* are observations about service involvement in a system transaction, plus pass/fail results about the transaction [Gonzalez-Sanchez *et al.*, 2011]. Involvement of the services in a transaction is termed a *spectrum*. The spectra of several transactions form an activity matrix, with a binary involvement vector per service, plus a binary outcome vector. Diagnosis is calculated by applying a similarity coefficient (SC) to each service activation vector and the outcome vector. The SC denotes the likelihood of a service being the faulty one, and, therefore, determines its position in the ranking.

The Ochiai similarity coefficient (SC_o) has been found to perform best in the past [Abreu *et al.*, 2006], however, more recently, feature selection approaches, i.e. Relief feature selection (FS_r), have been proposed as an alternative [Roychowdhury and Khurshid, 2011], to outperform SC_o in some cases. Whereas SC_o compares service involvement vectors with the output vector, FS_r takes a different approach. It treats each spectrum as a vector of features leading to two distinct system transaction classes, i.e. pass/fail determined by the output vector. Then, it finds the nearest-hit and nearest-miss spectra for each spectrum based on the Euclidean distance. Last, it calculates the weight for each feature by summing up the differences between the distance to nearest-miss and to nearest-hit caused by the feature over all spectra. The feature of the heaviest weight, i.e. most relevant to the failed-class and least relevant to the passed-class, represents the diagnosis.

Because FS_r is more heavy-weight than SC_o, which is relevant in online diagnosis, the goal of this paper is to compare the performance of the two methods in terms of diagnostic accuracy in a service-based system, i.e. correctly and uniquely localizing the faulty service(s). This brings us to our two research questions:

RQ1: How do Ochiai and Relief compare in diagnosing a service-based system?

RQ2: Are there circumstances under which the usage of the heavier-weight Relief approach is justifiable?

The main contribution of this paper is the assessment of both approaches under a multiple-fault assumption in a service-oriented context. Even though software is deterministic, intermittency can occur in service-based systems. This comes from the mismatch between the observation granularity, i.e. service granularity vs. the granularity of the fault activation, i.e. instruction level granularity. In other words, even though a faulty service may have been used in a system transaction, the faulty statement may not have been activated, leading to pass observations when the faulty service was involved. Further contributions are the simulation of service-based systems in order to collect many experimental results with different system configurations, and a case study involving a real service-based system.

2 Background and Approach

Spectrum-Based Fault Localization. SFL calculates a diagnosis from observations about component

involvement in system transactions and pass/fail outcome of a transaction [Gonzalez-Sanchez *et al.*, 2011]. Component involvement is expressed in terms of block-hit spectra, representing which component was activated in a transaction [Reps *et al.*, 1997][Zoeteweij *et al.*, 2007]. Execution of system transactions adds spectra to a so-called activity matrix, and a binary verdict (pass=0, fail=1) to the output vector. The diagnosis is calculated by applying a similarity coefficient (SC) to each component's activation vector and the output vector. It denotes the likelihood of a component to be the faulty one, and determines the position of the component in the diagnosis ranking.

Ochiai SC. Any binary similarity coefficient may be used in order to produce a diagnosis, however, the Ochiai SC (SC_o) was found to work best in experiments [Abreu *et al.*, 2006]. SC_o is determined for each component activation vector (a_i) and the output vector (o_i), and it is based on three counters $n(1,1)$, $n(1,0)$, and $n(0,1)$ representing the respective numbers of occurrences that a_i and o_i form these combinations: $SC_o = n_{11}/\sqrt{(n_{11}+n_{01})\cdot(n_{11}+n_{10})}$.

Relief FS. As alternative to SC, feature selection (FS) for two classes was proposed. The Relief FS algorithm (FS_r) [Kira and Rendell, 1992] aims to select a set of features which are more relevant to separate the two classes, i.e. maximizing the difference between two classes. This is similar to SFL, which tries to find a set of components being more likely to cause the failed spectra, i.e. determining a spectrum to fail or pass. Which of the features is more relevant is determined by a relevance value W for each feature f_i, i.e. each component in the spectra; $W_i = 1/N \sum_{k=1}^{N}(diff(X_k[i], X_k^m[i])^2 - diff(X_k[i], X_k^h[i])^2)$, with the two *diff*s representing the distances of a sample (X_k, i.e. spectrum) to its nearest miss (X_k^m) and nearest hit (X_k^h) samples at the ith position, respectively. X_k^h or X_k^m is the sample having the minimum Euclidean distance to X_k among all samples of the same or different class with X_k. The difference between two *diff*s represents how much f_i can differentiate X_k from the other class. N represents the total number of observations (spectra). Features are ranked with their relevance to see which features have the largest relevance, representing the most relevant indicators for two classes, i.e. which components are more critical to the result of a spectrum.

In a direct comparison using the Siemens set , FS_r was found to outperform SC_o in some cases [Roychowdhury and Khurshid, 2011]. Because FS_r is considerably more heavyweight than SC_o, in this paper, we study under which circumstances, FS_r may also present a tradeoff over SC_o in service-based systems.

SFL for Services. The application of SFL in service-based systems requires adaptations in terms of the component granularity, system activation, component coverage and verdicts [Chen *et al.*, 2012]. The natural choice for the component granularity is the service or the service operation. Service involvement in transactions cannot be observed through coverage tools, since services are activated from different execution contexts. In service-based systems, activation information can be produced through associating transaction IDs with the

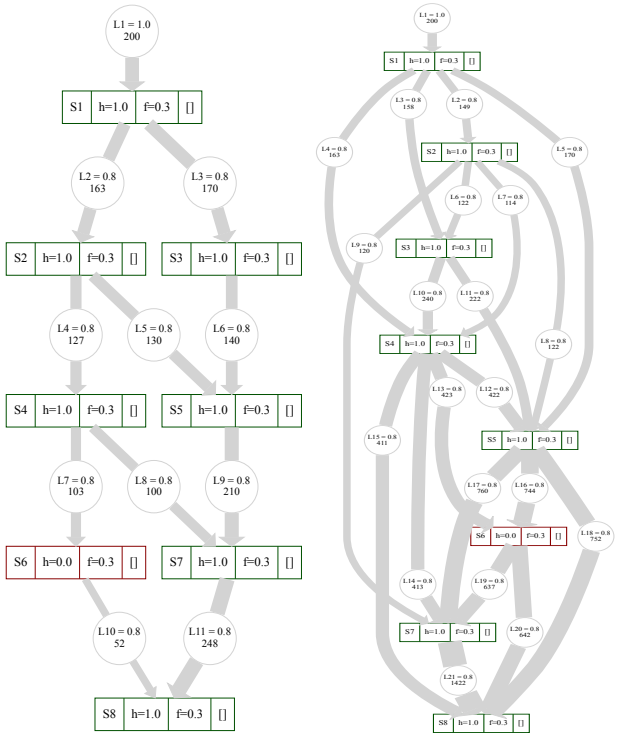


Figure 1: Simulated topologies for illustration only

services processing a transaction. Verdicts in service-based systems can be realized through dedicated monitors that log message contents and exceptions.

3 System Simulations

SFL Simulator. Before delving into a fully fledged case study with a service-based system in Sect. 4, we performed an initial assessment with our SFL Simulator[1]. It provides functions for setting up component topologies, executing the topologies thereby gathering coverage information, and calculating diagnoses. A topology is created by defining a number of components. Each component is defined by the component name, health, and failure probability. Health denotes the probability that a component will not produce an error. Failure probability determines the likelihood of a component to issue a failure immediately when a fault is propagated to it or its own fault is activated. Once a fault is activated it is checked in every subsequent component invocation whether or not it leads to failure. If a failure is detected, the execution is stopped. If all component failure probabilities are set to 0.0, the error is detected at the end of the execution. Components in a topology can be connected through defining an invocation link between them with an associated invocation probability. This defines the likelihood that a linked component will be invoked during execution.

Based on the topology with components and invocation links, the simulator can be controlled to perform executions. This requires that one or several entry points (components or links) are activated. Every activation of the topology leads to a particular control flow according to the initially defined probabilities, thereby generating coverage observations for the activity matrix and pass/fail information. These observations are collected and used to calculate a diagnosis ranking.

[1] https://github.com/SERG-Delft/sfl-simulator

Simulation Setup. We performed simulations with seven basic topologies, two of which are illustrated in Fig. 1. Both are comprised of 8 components each (i.e. the boxes) with links between them (i.e. circles and arrows). Topology A has a lower number of invocation links between the components (11 links), and topology B shows a higher number of links (22 links), leading to more diverse component activation paths. The other topologies used have a varied number of components and links between them. Initial simulations hinted at diagnosis performance differences according to the number and type of links between components. In the experiments, the failure probabilities of the components were varied in order to simulate different failure detection capabilities of the systems. The health value of the faulty component(s) was varied in order to simulate intermittent failure behavior, i.e. components only fail sometimes when activated. The link probabilities between the components were also varied in order to simulate different invocation patterns in the systems. Finally we tried different number of faulty components (1, 2, and 3 faults) in a topology, and looked at different numbers of observations per experiment, i.e. 20, 50, 100, and 200 spectra in order to test whether the number of pass/fail observations would make any difference. The performance of SC_o and FS_r is measured in terms of percentage of the number of faults that are correctly and uniquely identified by each method, against the total number of faults, i.e. how many of the diagnoses lead to correct and unambiguous results.

Simulation Results. In total, we performed 4500 simulations with different topologies and topology settings. The activity matrices produced during the simulations were fed into the respective ranking methods and the results are compared. In the interest of space we cannot present all results, so we concentrate on, as far as we are concerned, the most relevant findings.

Single-fault assumption: In the single fault case, both methods produce comparable results when the health of the faulty component is set to 0.0 (100% faulty), i.e. every activation of the faulty component leads to a failure. If the health of the faulty component is increased, but the number of observations stays low (i.e. 25 in our experiment), FS_r performs better than SC_o in topologies with very low link probabilities (<=0.3). However, Ochiai performs better than Relief when the link probabilities are increased (>0.3) (top chart in Fig. 2), although, the differences between the two techniques are minute. Low link probability results in high filtering of invocations in a topology, i.e. it is likely that many components in a spectrum are not activated. This leads to activity matrices that are full of zeroes, and it may hint to a potentially better performance of Relief in such cases. Furthermore, when both the health of the faulty component and the total number of observations are increased, the performance of FS_r is improved towards the SC_o's, although SC_o is still slightly better than FS_r (middle chart in Figure 2).

Multiple-fault assumption: When the number of faulty components in the system is increased (up to three faults), the performance of the two methods changes: FS_r always outperforms SC_o if the health of all faulty components is set to 0.0. When the health

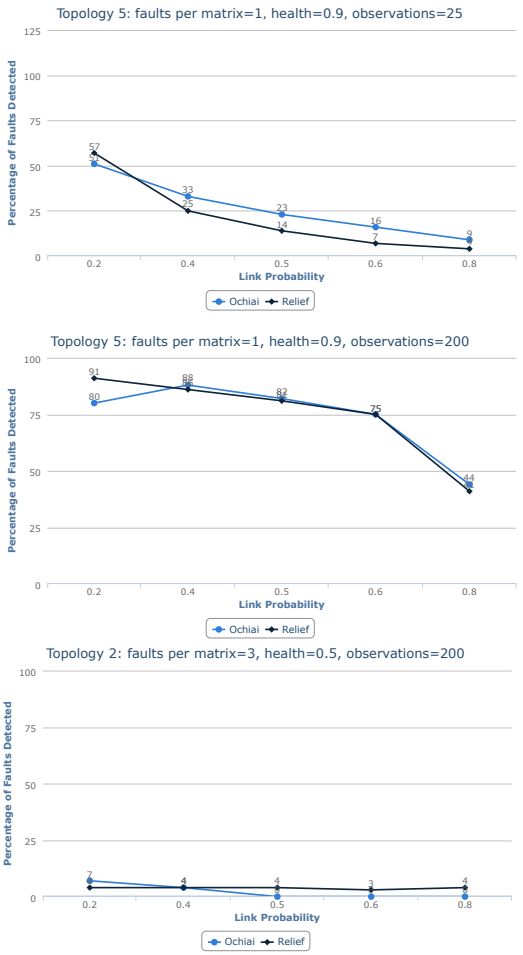


Figure 2: Example charts comparing Ochiai and Relief

probabilities are increased, SC_o performs slightly better than FS_r when the link probabilities are very low (<0.4). However, FS_r outperforms SC_o as the link probabilities are increased (bottom chart in Fig. 2). Higher link probability leads to more component activations in a topology, and this can further cause the tight interaction between components to happen more often. It hints at FS_r outperforming SC_o in tightly-coupled topologies for diagnosing multiple faults.

We also observed an extreme phenomenon that FS_r failed completely regardless of the link probability in some topologies for two-fault experiments, while SC_o performs slightly better. After careful investigation, we found that the two faulty components are always involved together in the failed spectra in those incorrect-diagnosed activity matrices, i.e. tight interaction between the two faulty components. Such case may imply that FS_r is not able to identify faults in the matrices where each fault is not separately activated.

In general, our simulations suggest that SC_o is more robust in the single-fault case, and that FS_r performs better in the multiple-fault case. However, the diagnosis performance for the two methods is affected by many factors, such as the link probabilities, the number of faulty components, their health state, their interactions, and the number of spectra, etc.

4 Case Study

After having compared the diagnosis capability of Relief and Ochiai in simulations in order to get an ini-

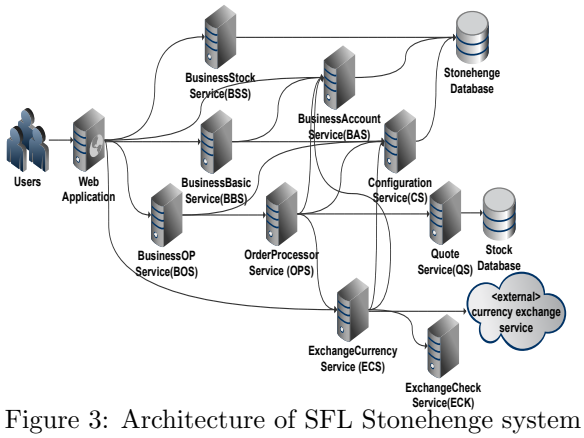


Figure 3: Architecture of SFL Stonehenge system

tial idea under which circumstances one or the other will be more beneficial, we also conducted an experiment with our original case study SFL Stonehenge introduced in [Chen *et al.*, 2012].

SFL Stonehenge is a service-based system simulating the stock market. It is deployed to Ebay's open source service framework Turmeric.It supports users in buying and selling of stocks, checking orders, and performing currency conversion operations for foreign stock acquisition. Fig. 3 illustrates the system. It is comprised of 10 web services including one *external currency exchange service*, plus a *web application* for user interaction. In addition, it accesses two data stores. The services provide the following operations. *BusinessBasicService* (BBS) and *BusinessAccountService* (BAS) provide the functions for user authentication and login. *BusinessOPService* (BOS) and *BusinessStockService* (BSS) are used for buying and selling stock, checking orders, and compiling market summaries. *QuoteService* (QS) and *OrderProcessorService* (OPS) are used to process stock orders placed by a user. *ExchangeCurrencyService* (ECS) and *ExchangeCheckService* (ECK) are responsible for currency operations, and the *ConfigurationService* (CS) binds all the other services together and acts like a registry.

4.1 Conducting the Case Study

We applied the PIT mutation tool to create a total of 136 faulty versions of the services. Each faulty version of a service contains only one fault. For each experiment, a combination of healthy and faulty services is deployed in the Jetty web server, according to the number of faulty services to be diagnosed by the two approaches under consideration, i.e. SC_o and FS_r. Apache's JMeter is then used to perform 50 system transactions on the service configuration, in order to achieve a realistic coverage of all service operations and fault activations, representing activity matrices with 50 spectra per experiment. The coverage information of service operations for each transaction is logged through Turmeric's built-in monitoring framework. How this is done in detail is described in [Chen *et al.*, 2012]. Upon completion of all 50 system transactions, a diagnosis engine is invoked to parse the monitoring data, identify the failures in the system, and create an activity matrix with an output vector. Finally, the diagnosis engine applies both Ochiai and Relief algorithms, in order to calculate the ranking of service operations based on the same activity matrix and output vector. We are now able to compare the diagnosis results from Ochiai and Relief.

Table 1: Diagnosis Results for 1-fault case

Service	Faulty Versions	SC_o Inc	Amb	Cor	Cor %	FS_r Inc	Amb	Cor	Cor %
BAS	7	0	0	7	100%	0	0	7	100%
BBS	25	2	0	23	92%	1	0	24	96%
BOS	18	4	0	14	77.8%	4	0	14	77.8%
BSS	8	0	0	8	100%	0	0	8	100%
CS	7	0	0	7	100%	0	0	7	100%
ECK	8	0	0	8	100%	0	0	8	100%
ECS	22	19	0	3	13.6%	19	0	3	13.6%
QS	10	1	0	9	90%	1	0	9	90%
OPS	31	10	0	21	67.7%	8	0	23	74.2%

4.2 Case Study Results

Single-fault assumption. Table 1 summarizes the diagnosis results for experiments performed under the single-fault assumption. That is, the system only contains one faulty service per experiment. The table shows for each *service* the number of *faulty versions* generated by the PIT mutation tool, summing up to the total of 136 experiments. It means each system configuration includes a faulty version of one service, and healthy versions of all other services. Then it shows for each of the two considered diagnosis approaches, Ochiai (SC_o) and Relief (FS_r), the number of incorrect (*Inc*), ambiguous (*Amb*), and correct (*Cor*) diagnoses. In addition, the percentage of the correctly performed diagnoses is shown for easy comparison. A *correct diagnosis* ranks only the single faulty service as top component. An *ambiguous diagnosis* ranks the faulty service at the top, but includes other healthy services under the same highest rank. An *incorrect diagnosis* ranks any arbitrary healthy service higher than the faulty one.

The results shown in Table 1 indicate that, in the single-fault case, Ochiai and Relief deliver comparable diagnostic performance in terms of ranking the faulty services correctly. For seven out of the nine services used in the case study, both approaches show identical outcomes. For two faulty services, BBS and OPS, Relief performs slightly better than Ochiai, i.e. in one and two experiments, respectively.

Multiple-fault assumption. Tables 2 and 3 summarize the diagnosis results for experiments performed under multiple-fault assumption. Table 2 contains the diagnosis results for experiments performed with two faulty services. In this case, we chose *BAS* and *BSS* as faulty services, i.e. each configuration per experiment contains one faulty version of *BAS* and one faulty version of *BSS*, all other services are healthy, because both Ochiai and Relief are able to achieve 100% correct diagnoses in the single-fault experiments for these services. That way, we can achieve a fair comparison.

The assessment criteria are also based on correct, ambiguous and incorrect diagnosis, as for the single-fault scenario. However, in order to achieve an absolute comparison between the two diagnosis ranking approaches, Ochiai and Relief, we have to look at finer-grained results, also in terms of combinations of correct, and ambiguous. Thus, in addition to the total number of diagnoses, i.e. 56, Table 2 shows the cases that both faulty services are correctly diagnosed (*2 Cor*); one is correctly diagnosed and the other one is ambiguous (*1 Cor + 1 Amb*); two are ambiguous (*2 Amb*); only one is correctly diagnosed (*1 Cor*); only one is ambiguous (*1 Amb*); and both faulty services are incorrectly diagnosed (*2 Inc*). The last value indicates the percentage

Table 2: Diagnosis Results for 2-fault case

	Tot	2 Cor	1 Cor + 1 Amb	2 Amb	1 Cor	1 Amb	2 Inc	Cor %
SC_o	56	41	8	0	7	0	0	73.2%
FS_r	56	3	0	0	53	0	0	5.4%

of the ranking techniques (SC_o or FS_r) identifying both faulty services correctly and unambiguously (2 Cor) out of the total number of diagnoses.

The numbers shown in Table 2 clearly indicate that Ochiai performs reasonably well, in 73.2% of the cases, in correctly pinpointing both faulty services, whereas Relief performs really poorly (only 5.4% of the cases). Both techniques do not fail completely, indicated through the fact that none of the diagnoses are completely incorrect (*2 Inc* = 0).

Table 3 contains the diagnosis results for the three-fault case. We chose *ECK* as the third faulty service, another one that can always be correctly diagnosed in the single-fault case by both techniques. The rest of the setup corresponds to the 2-fault case, except for the fact that we include more combinations of correct and ambiguous diagnoses. *3 Cor* represents the number of diagnoses in which all three components have been identified as the faulty ones by a technique, *2 Cor + 1 Amb* refers to two correctly identified faulty components and one ambiguous one, and so on.

Out of 100 experiments performed in total, Ochiai is still able to identify the three faulty services correctly in 29 cases. In 12 cases, it identifies two faulty services correctly, with one service being ranked high, but together with other healthy services (*2 Cor + 1 Amb*). In contrast, Relief fails entirely on the 3-fault case. It pinpoints 2 faulty services correctly, as in the 2-fault case, and is able to identify many single faulty services correctly. This is a clear indicator that Ochiai outperforms Relief considerably under multiple-fault assumption.

5 Discussion

General Observations. We have assessed the diagnostic accuracy of FS_r and SC_o through simulations and a case study. The general results from the two studies seem quite contradictory, however, further investigation indicates that the case study results confirm most findings from the simulations.

One of the major differences between the simulations and case study is the diversity of the diagnosed systems. Although our case study represents a realistic service-based system, when comparing it to the simulations, its system settings are limited, e.g., the link probability has less variation. Moreover, we use the same set of input data to invoke all services for each experiment, which results in each diagnosis having the same activity matrix, albeit with different output vectors. However, in the simulations, the activity matrices differ due to the random nature of the simulator. We now discuss the simulation and case study results in more detail.

Observation number: In the case study the inputs to the system remain the same; we thus fixed the number of observations at 50. For the simulations, we made 25, 100 and 200 observations for each topology configuration (component health, link probabilities).

Table 3: Diagnosis Results for 3-fault case

	Tot	3 Cor	2 Cor + 1 Amb	1 Cor + 2 Amb	2 Cor	1 Cor + 1 Amb	1 Cor	3 Inc	Cor %
SC_o	100	29	12	13	10	5	31	0	29%
FS_r	100	0	0	0	3	0	97	0	0%

Component activation: Since it is hard to determine the overall link probability for our case study system, due to the diverse interactions between services, we check the service activations in the activity matrices. Each matrix has 34 rows, representing the service interfaces, and 50 columns, representing 50 transactions. Of these 1700 combinations (i.e., 34×50), there are only 117 component activations (6.88% of the total possible). In contrast, the various simulation runs cover the results for each topology with various link probabilities, and the link probability is the likelihood that the linked component will be activated during execution. Therefore, low link probabilities in the simulation also results in a low number of component activations. For example, when the link probability is 0.2, the percentage of the component activation in the matrix is around 20%.

Fault intermittency: The case study contains 136 faulty versions of services, and only some of the faults showed intermittency during the experiments. In the case study results for single fault, both FS_r and SC_o only failed in diagnosing the system having fault intermittency. However, in the results for multiple faults, FS_r also failed when the system does not have fault intermittency. This is the opposite of the simulation results, where FS_r always outperforms SC_o in multiple fault cases with no fault intermittency. One possible reason for this mismatch is that the case study uses the same inputs and has a low number of observations which may fool Relief [Dash and Liu, 1997])

Based on the above analysis, we can outline a set of findings from the simulations confirmed by the case study: 1) FS_r and SC_o have comparable performance when diagnosing systems with a single fault and no fault intermittency; 2) FS_r performs slightly better than SC_o when diagnosing a single fault system with fault intermittency, with low component activation and low number of observations; 3) SC_o outperforms FS_r when diagnosing a multiple-fault system with fault intermittency, with low component activation and low number of observations.

Threats to validity. There are a number of threats that might invalidate our findings. One threat is about the Relief algorithm, which does not consider a particular situation when a sample may not have nearest-hit/nearest-miss samples. However, in the experiments under the single fault assumption, it is very much possible that the whole experiment contains only one failed test, which means this test does not have its nearest-hit neighbor. For this situation, when calculating weights, we simply set the difference between the failed test and its nearest-hit to zero ($diff(X, X^h) = 0$), the minimum value. Although this applies to all components when calculating their weights from the failed test, without a thorough validation, we are not completely sure that this setting does not cause any disturbance to Relief.

Another threat is that we did experiments under the multiple-fault assumption only with services which receive 100% correctness of diagnosis in the single-fault case. Such a basic set-up excludes unexpected issues when diagnosing service systems.

6 Related Work

[Roychowdhury and Khurshid, 2011] conducted a study to apply FS techniques to SFL, and compared the diagnosis capability of Relief with Ochiai and other algorithms. In their research they only assessed those algorithms in the single-fault case with the Siemens Test Suite, while our work explores the diagnostic performance of Ochiai and Relief for both single and multiple faults in a service-oriented context. In addition, the assessment metrics used are different: they check the percentages of examined code before the fault is localized for each algorithm, whereas we compare how many times the algorithms are able to correctly and uniquely pinpoint the faulty component(s).

[Abreu *et al.*, 2007] investigated the influences of various SC algorithms in SFL, and confirmed that Ochiai achieves highest accuracy in diagnosing single-site faults. Their work only considers the single fault case and FS algorithms are not included. [Dash and Liu, 1997] evaluated a number of FS algorithms for classification, and concluded that Relief is effective for large datatsets and it tolerates noise better than other FS algorithms.

There are some studies on statistics-based diagnosis for software systems. For example, [Wong *et al.*, 2010] propose a number of code coverage-based heuristics to localize faults; [Renieris and Reiss, 2003] present a nearest-neighbor fault localization approach that selects a successful run which is most similar to the failed run and compare two runs in order to find the fault.

7 Conclusion

In this paper, we have compared the diagnostic accuracy of SC_o and FS_r under the service-oriented context. Following we address our research questions:

RQ1 *–How do Ochiai and Relief compare in diagnosing a service-based system?* We have conducted 4500 simulations with various topologies and topology settings, and a case study with a realistic service-oriented system, in order to compare the diagnostic performance of SC_o and FS_r. Both simulation and case study results confirm that (1) FS_r and SC_o have equal performance when diagnosing single-fault service systems with no fault intermittency; (2) FS_r performs slightly better than SC_o when the diagnosed system has fault-intermittent behavior, low component activation and low observation number, for diagnosing single fault; (3) SC_o outperforms FS_r when the diagnosed system has the similar settings to (2), but for diagnosing multiple faults. In addition, the simulation results also indicate that SC_o outperforms FS_r in service systems with link probabilities higher than 0.3, for diagnosing single fault, and that FS_r outperforms SC_o when diagnosing similar systems for multiple faults.

RQ2 *–Are there circumstances under which the usage of the heavier-weight Relief approach is justifiable?* From the simulation and case study results, we observed that FS_r outperforms SC_o when diagnosing single fault in the system with fault intermittency, low link probability (<0.3) and low observation number, and when diagnosing multiple faults in the system with fault intermittency and high link probability (>0.4). In addition, we also noticed that in general the diagnosis performance of FS_r can be improved when the number of observations in the simulations are increased. Although the simulations show that FS_r considerably outperforms SC_o when diagnosing multiple faults in the system without fault intermittency, the case study results are against this. It may be due to the limited information in the activity matrices produced by the case study. This implicitly indicates that more and more diverse data can benefit FS_r.

To sum up, based on some typical features of a service system may affect diagnosis, such as loosely coupling and fault intermittency (due to the mismatch between fault activation granularity and monitoring granularity), we would like to propose that (1) When diagnosing single fault in a very loosely-coupled service system, FS_r is more suitable than SC_o, while SC_o is more lightweight. (2) When diagnosing multiple faults in a very loosely-coupled service system and the service activities are relatively limited, SC_o is better. (3) FS_r can achieve high diagnostic accuracy in the service system offering highly diverse activities, but as a trade-off is more calculation-intensive.

Acknowledgements: Thanks to NWO for sponsorship, and our industrial partners Adyen and Exact.

References

[Abreu *et al.*, 2006] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proc. Int'l Symp. on Dependable Computing*, pages 39–46. IEEE, 2006.

[Abreu *et al.*, 2007] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund. On the accuracy of spectrum-based fault localization. In *TAICPART-Mutation*, pages 89–98, 2007.

[Canfora and Di Penta, 2009] G. Canfora and M. Di Penta. Service-oriented architectures testing: A survey. In *Software Engineering*, pages 78–105. Springer, 2009.

[Chen *et al.*, 2012] C. Chen, H.-G. Gross, and A. Zaidman. Spectrum-based fault diagnosis for service-oriented software systems. In *Proc. Int'l Conf. on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012.

[Dash and Liu, 1997] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1:131–156, 1997.

[Gonzalez-Sanchez *et al.*, 2011] A. Gonzalez-Sanchez, R. Abreu, H-G Gross, and A.J.C. van Gemund. Spectrum-based sequential diagnosis. In *Int'l Conf. on Artificial Intelligence*, pages 189–196, 2011.

[Kira and Rendell, 1992] K. Kira and L.A. Rendell. A practical approach to feature selection. In *Proc. Int'l workshop on Machine learning*, pages 249–256, 1992.

[Renieris and Reiss, 2003] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *Int'l Conf. on Automated Softw. Eng.*, pages 30–39, 2003.

[Reps *et al.*, 1997] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In *ESEC/FSE*, pages 432–449. Springer, 1997.

[Roychowdhury and Khurshid, 2011] S. Roychowdhury and S. Khurshid. Software fault localization using feature selection. In *Int'l Workshop on Machine Learning Technologies in Softw. Eng.*, pages 11–18. ACM, 2011.

[Wong *et al.*, 2010] W. Wong, V. Debroy, and B. Choi. A family of code coverage-based heuristics for effective fault localization. *J. of Syst. and Softw.*, 83(2):188–208, 2010.

[Zoeteweij *et al.*, 2007] P. Zoeteweij, R. Abreu, R. Golsteijn, and A.J.C. van Gemund. Diagnosis of embedded software using program spectra. In *Proc. Int'l Conf. and Workshops on Engineering of Computer-Based Systems (ECBS)*, pages 213–220. IEEE, 2007.

A Learning Anomaly Detection Algorithm for Hybrid Manufacturing Systems

Oliver Niggemann[1,2] and **Asmir Vodenčarević**[1] and **Alexander Maier**[2] and **Stefan Windmann**[1] and **Hans Kleine Büning**[3]
[1]Fraunhofer Application Center Industrial Automation, Lemgo, Germany
e-mail: {oliver.niggemann, stefan.windmann, asmir.vodencarevic}@iosb-ina.fraunhofer.de
[2]Institute Industrial IT, Lemgo, Germany
e-mail: {alexander.maier}@hs-owl.de
[3]University of Paderborn, Paderborn, Germany
e-mail: {kbcsl}@upb.de

Abstract

For complex and distributed technical systems, model-based anomaly detection solutions often show better results than approaches, which do not use explicit behavior models. But so-far, the creation and maintenance of such models turned out to be the major problem of such model-based approaches. One solution to this problem is provided by machine learning: Behavior models can be learned automatically based on system observations. Such a machine-learning-based solution would create an anomaly detection algorithm, which could learn and therefore could adapt itself to new situations.

In this paper, such an algorithm is described: The ANODA algorithm for anomaly detection uses the HyBUTLA algorithm to learn behavior models in form of hybrid timed probabilistic automata. For the first time, a thorough theoretical analysis of this algorithm is presented. Practical results also underline the applicability of these algorithms.

1 Motivation

Cyber-physical systems and intelligent technical systems are a major trend in the fields of industrial automation. The common denominator of all such projects is the integration of more cognitive capabilities in the automation systems. Examples for such capabilities are the self-diagnosis or the self-optimization of a plant's energy consumption.

At the heart of such approaches lies a model of the system behavior. Such a model can be used to detect discrepancies between expected and observed behavior. To this aim, two major questions need to be answered: (*1*) *Which model formalism can capture the behavior of a typical production plants?* and (2) *How can these models be learned automatically from system measurements?*

These questions are important since a manual creation of such models is hardly feasible: The efforts are too high, experts are scarce and systems are too complex. To answer the first question, model formalism for hybrid production systems is presented. This paper focuses on the learnability of this model formalism. The paper then applies the novel algorithm HyBUTLA for automated generation of behavior models. This approach is finally verified using several model-learning experiments and anomaly detection scenarios.

Generally speaking, two classes of algorithmic approaches exist for the detection of anomalous situations:

I. Phenomenological Approach: Here, the system output including its sensors, data and its energy consumption is directly classified as correct or anomalous.

II. Model-based Approach: In order to detect anomalies automatically, a model-based approach can be used. A model is used to simulate the normal behavior of a plant. For this, the simulation model needs all inputs of the plant, e.g. product information, plant configuration, plant status, etc. If the actual measurements vary significantly from the simulation results, the behavior is classified as anomalous.

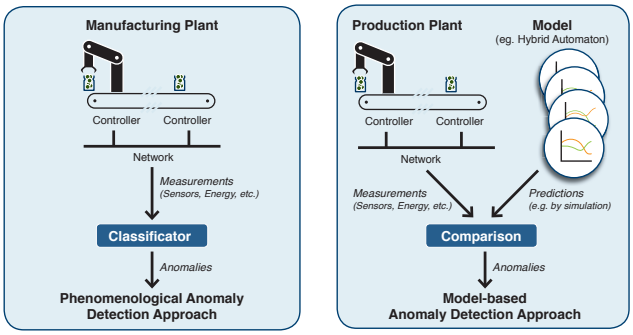


Figure 1: Phenomenological vs. Model-Based Approach

While phenomenological approaches are often more straight-forward and do not require a system model, they have one major inherent drawback: They must deduce against the direction of causality since they deduce from measurements (i.e. symptoms) to anomalies. For complex distributed systems with many interdependencies between components and complex causalities, this is a hard task because a high number of classification rules is needed. Since the analysis of a plant behavior depends on a large number of input data and normally deals with distributed plants and automation systems, a model-based approach is chosen here.

Theoretically, engineers are able to implement model-based anomaly detection solutions manually in the automation systems, but only at a high price: For each plant, a model must be developed manually. With growing plant complexities and shorter plant construction times, this approach is rendered unrealistic. The solution for this challenge are learning anomaly detection algorithms in the automation systems. This is mainly realized based on observations of the plant behavior—in addition to a-priori knowledge about the plant and the underlying physical processes.

2 Hybrid Model: Expressiveness vs. Learnability

2.1 Hybrid Manufacturing Systems

Hybrid manufacturing systems are complex systems that are characterized by four interacting behavioral types, namely:

Discrete behavior: Manufacturing process typically consists of several modes of operation, which are driven by a discrete control system. When the process is in one mode of operation, it is said to be in one state.

Continuous behavior: In addition to state-based discrete behavior, hybrid systems are in general dependent on changes of the process variables (e.g. pressure or temperature), which change continuously over time.

Dynamic behavior: The behavior of the whole system is defined by both the changes in control signals and process variables. These changes happen within certain time intervals, and therefore is the system behavior time-dependent.

Stochastic behavior: The system behavior is also influenced by a number of random factors, such as an external disturbance, a process or measurement noise, or a human factor. As a result, all system changes happen with certain probability.

Various formalisms exist that can model aforementioned characteristics of a hybrid manufacturing system (e.g. hybrid bond graphs [Narasimhan and Biswas, 2007] or hybrid Petri nets [David and Alla, 2001]). In this paper we focus on hybrid timed automata [Alur *et al.*, 1995; Henzinger, 1996].

2.2 Hybrid Timed Automata

Hybrid automata are a popular formalism for modeling hybrid system. Here we also model probability and timing information about the system.

An example of a hybrid timed automaton is given in Figure 2. It shows two states, namely *run* and *stop* and a transition between them that is triggered by an event a. Typically, such event is triggered by the control system and initiates some change in a system, e.g. opening of a valve. Both timing constraint $t < 20ms$ and a probability $p = 0.2$ are associated with the transition. They respectively show in what time range and with what probability the corresponding transition can be triggered. The behavior of continuous process variables relevant for every given state is defined by a θ function.

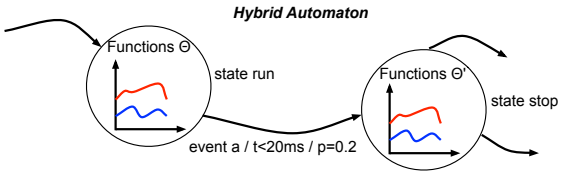


Figure 2: Example of a hybrid automaton.

Formally, the hybrid automaton used here is defined as follows:

Definition 1. *(Hybrid Timed Automaton) A hybrid timed automaton is a tuple* $A = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$*, where*

- S *is a finite set of states,* $s_0 \in S$ *is the initial state, and* $F \subseteq S$ *is a set of final states,*
- Σ *is the alphabet comprising all relevant events.*
- $T \subseteq S \times \Sigma \times S$ *gives the set of transitions. E.g. for a transition* $\langle s, a, s' \rangle$*,* $s, s' \in S$ *are the source and destination states and* $a \in \Sigma$ *is the trigger event.*
- *A set of transition timing constraints* Δ *with* $\delta : T \rightarrow I, \delta \in \Delta$*, where* I *is the set of time intervals.*
- *A function* $num : T \rightarrow \mathbf{N}$ *counts the number of observations that used the transition.*
- *A single clock* c *is used to record the time evolution. At each transition, the clock is reset. This allows only for the modeling of relative time steps.*
- *A set of functions* Θ *with elements* $\theta_s : \mathbf{R}^n \rightarrow \mathbf{R}^m, \forall s \in S, n, m \in \mathbf{N}$*. I.e.* θ_s *is the function computing signal value changes within a single state* s*. I.e. several input and output variables are supported.*

The HyBUTLA algorithm for learning hybrid timed automata was given in [Niggemann *et al.*, 2012] and briefly described in the following section. It is important to emphasize that this algorithm learns both model parameters and a model structure.

3 Phase 1: Learning the Model

Since even the learning of the simplest well-known deterministic finite automata (DFAs) from given finite data [Gold, 1978] is proven to be NP-complete, we focused our attention on so-called *learning in the limit* [Gold, 1967]. In this learning framework, the learner is given more and more data until the convergence to some target automaton is achieved. Our application area imposes one additional, stricter requirement: learning from *positive* data only. In computational learning theory, one can learn either from positive and negative data, or just from positive data. Data are positive with respect to some finite state machine, when they can be generated by that machine. Data taken in manufacturing systems (i.e. system logs) are dominantly positive. Table 1 shows the main characteristics of hybrid timed automata expressiveness that influence their learnability. The results are generalized from the existing results for simpler DFAs and deterministic timed automata [Verwer *et al.*, 2008]. Looking at this table, it is clear that our hybrid timed automaton model has to be deterministic, stochastic and has to have only one clock for modeling time. Its formal definition was given in the previous section.

Table 1: Expressiveness vs. learnability of hybrid timed automata.

Expressiveness characteristic	Learnable in the limit from positive data
non-determinism	No [de la Higuera, 1997]
stochastic	Yes [Vodenčarević *et al.*, 2011]
multiple clocks	No [Verwer *et al.*, 2008]
single clock	Yes [Verwer *et al.*, 2009]

Learning the hybrid model consists of two steps, which are described in this section. First, the underlying finite state machine is learned from observations. In the second step, the continuous functions Θ for the particular modes of the state machine are learned.

Step 1: Learning the finite state machine

The algorithm works as follows: First in **step (0)**, all relevant signals are measured during a system's normal operation and stored in a database. This measurement approach

Algorithm HyBUTLA (Σ, $\mathcal{O}$):
Given:
(1) Events Σ
(2) Observations $\mathcal{O} = \{\mathbf{O}_0, \ldots, \mathbf{O}_{n-1}\}$ where $\mathbf{O}_i \in (\Sigma \times \mathbf{R})^*$, $\mathbf{O}_i$ is one sequence of timed events (e.g. a system cycle)
Result: Hybrid Automaton $\mathcal{A}$
(1) Compute events Σ based on $\mathcal{O}$.
(2) Build prefix tree $PTA = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$ based on $\mathcal{O}$. Let S' be all non-leaf nodes in S.
(2.1) $\forall s \in S$ learn $\theta_s \in \Theta$ using continuous data from $\mathcal{O}$. PTA is a hybrid automaton.
(3) **for all** $v, w \in S'$ in a bottom-up order **do**
(3.1) **if** compatible(v, w) **then**
(3.1.1) $\mathcal{A}$ = merge(v, w)
(3.1.2) determinize($\mathcal{A}$) **od**
(4) **return** $\mathcal{A}$

Algorithm 1: Hybrid automata identification algorithm HyBUTLA.

for distributed plants is described in [Pethig and Niggemann, 2012].

In **step (1)**, events are generated. An event is generated whenever a discrete variable changes its value—this often corresponds to an actuator or sensor signal in a technical system such as switching a valve or the toggling of a photoelectric barrier. These signals are used later on to trigger transitions between states, i.e. these events define mode switches. Furthermore, more events may be generated whenever a continuous signal crosses a manually defined threshold. In **step (2)**, a prefix tree acceptor (PTA) is built. Such a PTA is a hybrid timed automaton in the form of a tree. In a PTA, each sequence of observations results in one path from the tree's root to a leaf; common prefixes of sequences are therefore stored only once.

The continuous behavior (i.e. the output process variables) within each state $s \in S$ is modeled in **step (2.1)** by learning the functions $\theta_s \in \Theta$ from definition 1. Details are given later in this section.

Now in **step (3)**, the compatible states are merged until a smaller automaton is reached, which can still predict the system behavior: First, in **step (3.1)**, the compatibility between two states is checked, which are chosen in a bottom-up order. The bottom-up order of the nodes is defined here by the lexicographic order of the shortest sequence of events leading to the node.

Two nodes are compatible if (*i*) probabilities of all corresponding transitions are similar, if (*ii*) the transition timings Δ are similar and if (*iii*) the nodes in their subtrees are also compatible according to the same compatibility criteria. Figure 3 outlines point (*i*) : The incoming transitions triggered by event a are similar if their probabilities are similar. These probabilities $\frac{num_1^i}{sum_1^i}$, $\frac{num_1^j}{sum_1^j}$ are computed based on the functions num from definition 1 where sum_1^i and sum_1^j are the sums of all num-function values of all incoming transitions of nodes v_i and v_j respectively. The outgoing transitions are handled in a similar manner.

Whether two probabilities $\frac{num_i}{sum_i}$ and $\frac{num_j}{sum_j}$ are significantly different is checked using the following Hoeffding test:

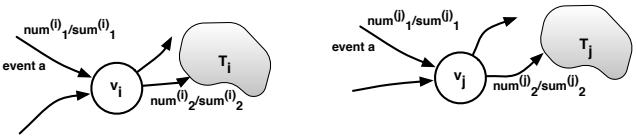


Figure 3: Compatibility of two nodes v_i and v_j

$$\text{fractions-different}(num_i, sum_i, num_j, sum_j) := \left| \frac{num_i}{sum_i} - \frac{num_j}{sum_j} \right| > \sqrt{\frac{1}{2} \log\left(\frac{2}{\alpha}\right)} \left(\frac{1}{\sqrt{sum_i}} + \frac{1}{\sqrt{sum_j}} \right)$$

where $(1-\alpha)^2, \alpha \in \mathbf{R}, \alpha > 0$ is the probability of the decision. If this inequality is true, the probabilities are different, i.e. states are different and will not be merged.

Concerning nodes point (*ii*) , two transition timings are similar, if the time intervals overlap. Finally, this compatibility is also checked for the nodes in all subtrees, e.g. subtrees T_i and T_j in Figure 3.

If the states are found to be compatible, they are merged in **step (3.1.1)**. When two states are merged, their portions of observed data are combined and new functions θ are learned. Because after the merging step the resulting automaton may be non-deterministic, the subtrees of the new merged state are made deterministic in step **(3.1.2)** by merging their nodes recursively (see also [Carrasco and Oncina, 1999] for details).

Step 2: Learning the continuous behavior in the modes of the hybrid automaton

The continuous variables Θ (e.g. energy consumptions) for all mode S of a hybrid automaton $A = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$ are here modeled as linear state space models as used e.g. by Kalman Filters.

Linear state space models are learned for each state of A. The continuous variables are described with a linear state space model (a standard random walk model):

$$x(k) = x(k-1) + u_s(k) \quad (1)$$
$$y(k) = x(k) + v_s(k) \quad (2)$$

where $x(k)$ denotes a value-continuous state variable e.g. describing the energy consumption. $u_s(k) \sim N(0; \sigma_s^{(u)})$ and $v_s(k) \sim N(0; \sigma_s^{(v)})$ denote Gaussian distributed process noise and measurement noise with standard deviations $\sigma^{(u)}$ and $\sigma^{(v)}$ respectively. The linear models are learned from a sequence of training data $y(k)$ which is observed. This data is collected in **step (2)** of the HyBUTLA algorithm and merged in **step (3)** (see Algorithm 1). Expectation-maximization (EM) is employed for parameter estimation (see [Ghahramani and Hinton, 1996]).

Details can be found in [Windmann *et al.*, 2013].

4 Phase 2: Model-based Anomaly Detection

In Algorithm 2, the anomaly detection algorithm ANODA is given (initially published in [Vodenčarević *et al.*, 2011]).

Three types of the anomalies can be detected using ANODA algorithm:

Improbable event sequences: This anomaly appears when the sequence of control signals (events) is incorrect (**step (2.1)** of Algorithm 2).

Given:
(a) Hybrid Automaton (HA) $A = (S, s_0, F, \Sigma, T, \Delta, Num, c, \Theta)$

(b) $o = (a, t, \mathbf{u}, \mathbf{y})$ is an observation from a system.
a is an event, t a point in time,
$\mathbf{u}$, $\mathbf{y}$ are the continuous input and output variables
ANODA algorithm receives such observations periodically.
(c) ε is a predefined alarm threshold for continuous signal
(d) $s_{curr} \in S$ the current state
Result: detected anomaly (if there exists one), otherwise "OK"
(1) $T' = \{e = (s_{curr}, a, s_{new}) \in T\}, s_{curr}, s_{new} \in S, a \in \Sigma$
(2) **if** $T' = \emptyset$ **then**
(2.1) **return** anomaly: unknown event error
else
(3) **if** $\exists e \in T' : t \in \delta(e)$ **then**
(3.1 $\mathbf{y}_p = \theta_{s_{new}}(\boldsymbol{u})$
(3.2) **if** $(\mathbf{y}_p - \mathbf{y}) > \epsilon$ **then**
(3.2.1) **return** anomaly: continuous signal error
(4) **else** anomaly: event timing error

Algorithm 2: Anomaly detection algorithm ANODA

Timing deviations: Such deviations are detected when an observed time deviates from learned time constraints (**step (4)** of Algorithm 2).

Incorrect continuous signal values: When deviations between observed and learned (expected) values of continuous signals are detected, this anomaly is signaled in **step (3.2.1)** of Algorithm 2.

For this, the algorithm first checks in step 1 which transitions T' match the observed signals. Step 3 then verifies whether the observed timing corresponds to one of the transitions in T'.

5 Theoretical Analysis of the Anomaly Detection Capabilities

In this section, we will for each aforementioned anomaly type compute the probability that the anomaly could not be found (false negative) or that an anomaly is identified incorrectly (false positive). Such errors can occur either because of incorrectly learned models or because of incorrectly classified measurements.

5.1 Errors with Improbable Event Sequences

Error Boundings

According to Hoeffding's inequality, we know that an observed transition probability $\frac{f}{g}$ is with a probability of $> (1-\alpha)$ at most $\sqrt{\frac{1}{2g}\log\frac{2}{\alpha}}$ away from the true probability p. I.e. with a probability of $< \alpha$, $|\frac{f}{g} - p| > \sqrt{\frac{1}{2g}\log\frac{2}{\alpha}}$.

Lemma 1. *For two observed transition probabilities $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ and one corresponding p (i.e. both observed transition probabilities are from the same distribution) it follows that with a probability of $< 2\alpha$ the following holds*

$$\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| > \left(\sqrt{\frac{1}{g_0}} + \sqrt{\frac{1}{g_1}}\right) \cdot \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}.$$

I.e. this is the probability that a correct merging is not done by the algorithms.

Proof.

$$\left|\left|\frac{f_0}{g_0} - p\right| - \left|\frac{f_1}{g_1} - p\right|\right| > \sqrt{\frac{1}{2g_0}\log\frac{2}{\alpha}} + \sqrt{\frac{1}{2g_1}\log\frac{2}{\alpha}}$$
$$\Leftrightarrow \left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| > \left(\sqrt{\frac{1}{g_0}} + \sqrt{\frac{1}{g_1}}\right) \cdot \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}$$

I.e. at least one of the two error sources $\left|\frac{f_0}{g_0} - p\right|$ or $\left|\frac{f_1}{g_1} - p\right|$ must be have exceeded the error limit. □

False positive errors for the anomaly detection occur when a correct transition is missing in the learned automaton. This can only happen if a correct merging of two states has been missed by the algorithm:

Theorem 1. *The probability of a false positive error in the anomaly detection p_{FP} is bounded by $O(\alpha n^3|\Sigma|)$ where n denotes the number of input samples.*

Proof. [1] According to lemma 1, the error for one call to the function *compatible* (see [Niggemann *et al.*, 2012]) is bounded by $O(\alpha|\Sigma|n)$ because for each node $2|\Sigma| + 1$ Hoeffding bounds are checked and each call to compatible may check recursively a maximum of $cn, c \in (0, 1]$ nodes (the PTA has a size of cn). Furthermore, a maximum of n^2 state pairs can be checked for compatibility[2]. □

False negative errors for the anomaly detection occur when an incorrect transition exists in the learned automaton. This can only happen if an incorrect merging of two states has been carried out by the algorithm:

So in the following we assume that two observed transition probabilities $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ stem from two different distributions. We now have to compute the probability that, besides being generated from different distributions, $\left|\frac{f_0}{g_0} - \frac{f_1}{g_1}\right| \leq \left(\sqrt{\frac{1}{g_0}} + \sqrt{\frac{1}{g_1}}\right) \cdot \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}$.

Lemma 2. *Let $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ stem from two different distributions and $\frac{f_0}{g_0} \to p_0, \frac{f_1}{g_1} \to p_1$. Then the probability of an incorrect merging is bounded by 2α iff $p_1 - p_0 > 2\gamma, \gamma = \max(\gamma_0, \gamma_1), \gamma_i = \sqrt{\frac{1}{2g_i}\log\frac{2}{\alpha}}$.*

Proof. Incorrect merging happens if either one of the observations $\frac{f_0}{g_0}, \frac{f_1}{g_1}$ lies within the intervals $[p_0 - \gamma_0], [p_1 - \gamma_1]$ of the other distributions. Figure 4 shows on its left hand side a situation whereat–correctly–a merging is not done.

On its right hand side the error scenario is shown, here we first assume that $\frac{f_0}{g_0}$ is incorrectly classified as stemming from p_1 and an incorrect merging happens.

In the left-hand scenario, it must hold that $p_1 - p_0 > \gamma_0 + \gamma_1$. And this inequality is true if $p_1 - p_0 > 2\gamma$.

Under this constraint, an (incorrect) merging can only happen if one of the two observations $\frac{f_i}{g_i}$ is outside its interval $[p_i - \gamma_i]$. The probability of this is (see the explanations above) 2α. □

[1] Picking up ideas from [Carrasco and Oncina, 1994].

[2] This assumes independent tests. De la Higuera and Thollard have shown in [Higuera and Thollard, 2000] that dependencies only lower the probabilities, i.e. the errors are still bounded as shown above.

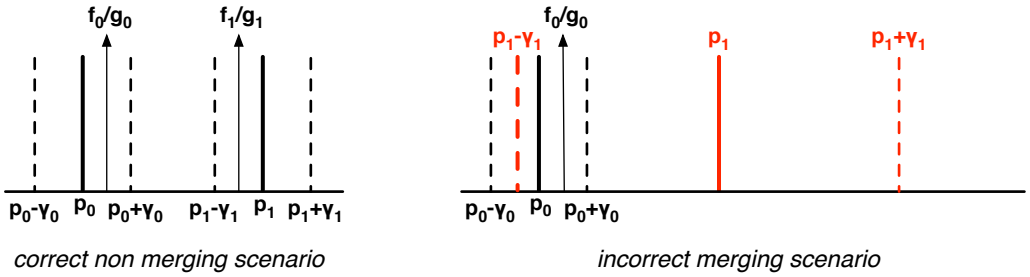


Figure 4: Idea for proving lemma 2.

Theorem 2. *If $\forall i,j : p_i - p_j > 2\gamma$, the probability of a false negative error in the anomaly detection p_{FN} is bounded by $O(\alpha n^3 |\Sigma|)$ where n denotes the number of input samples.*

Proof. According to lemma 2, the error probability for one test error for false negative errors corresponds to the error for a false positive error (iff $\forall i,j : p_i - p_j > 2\gamma$). I.e. we can repeat the idea of theorem 1. □

Remark 1. *The constraint $\forall i,j : p_i - p_j > 2\gamma$ means that we must assume that probabilities occur only in specific steps. And the step size depends on the number of samples in the least used transition. In practice this is no problem since the step size shrinks proportional to $\sqrt{1/n}$ if we presume that for each observation $\frac{f_i}{g_i}$ it holds that $g_i = cn, c \in \mathbf{R}$ where $n \in \mathbf{N}$ denotes the sample size.*

Corollary 1. *If $\forall i,j : p_i - p_j > 2\gamma$, both the false positive error p_{FP} and the false negative error p_{FN} (see theorems 1 and 2) can be kept below any given maximum error level p_{max} by choosing a specific value for α.*

Proof. If we set $\alpha = \frac{p_{max}|\Sigma|}{n^4}$, p_{FP} and p_{FN} will be below p_{max}. □

Learnability in the Limit

If the number of observations (including negative examples) is constant, Gold proved in [Gold, 1978] that the identification of a Deterministic Finite Automaton (DFA) of the given size $k \in \mathbf{N}$ is NP-complete, i.e. no efficient algorithms exist—under the assumption that $P \neq NP$. However, Oncina and Garcia showed in [Oncina and Garcia, 1992] that an efficient algorithm exist when the number of observations is not constant but can grow during an algorithm's runtime,—this is called identification in the limit. In this context, an algorithm is efficient when it has a runtime polynomial in the number of observations and the number of observations is polynomial in the size of the optimal automaton.

Definition 2. *(Identification in the limit) Let $\mathcal{O}$ be a set of observations for which an automaton $\mathcal{A}$ should be identified. An automaton learning algorithm identifies $\mathcal{A}$ in the limit if $\mathcal{A}$ is found based on observation $\mathcal{O}, |\mathcal{O}| > n$ for some $n \in \mathbf{N}$.*

It has been proven that stochastic DFAs [Carrasco and Oncina, 1999; Thollard *et al.*, 2000] and also stochastic deterministic timed automata with one clock [Verwer, 2010] are identifiable in the limit in polynomial time. Such stochastic automata use only positive observations.

For hybrid automata, the papers [Henzinger *et al.*, 1998; Alur and Dill, 1994] give decidability preconditions: *(i)* decoupled output variables, *(ii)* initialization after flow changes, *(iii)* one clock. These conditions are fulfilled here: *(i)* output variables are decoupled, *(ii)* Θ functions implicitly reinitialize variables, *(iii)* only one clock is used.

For the HyBUTLA algorithm, it can be said that as long as the Θ functions are approximated sufficiently well by some regression method in polynomial time, and as long as a single clock is used, HyBUTLA identifies the hybrid system behavior model in the limit in polynomial time. The polynomial runtime behavior has been proven above, so only the identification in the limit has to be shown:

Corollary 2. *If $\forall i,j : p_i - p_j > 2\gamma$, the algorithm HyBUTLA learns in the limit.*

Proof. Follows from theorems 1 and 2, if we e.g. set $\alpha = \frac{p_{max}|\Sigma|}{n^4}$, p_{FP} and p_{FN} will approximate 0 with growing n □

The concept of "learnability in the limit" has a crucial drawback: Nothing is said about the problem complexity in terms of data size n and in terms of runtime complexity. For this, two new concepts are introduced: 1. "Weak Polynomial Identification in the Limit with Probability One" requires that the algorithms has a polynomial runtime in the number of observations. 2. "Strong Polynomial Identification in the Limit with Probability One" additionally requires that only a polynomial number of observations in the automaton size is used.

Definition 3. *(Weak Polynomial Identification in the Limit with Probability One) Let $\mathcal{O}$ be a set of observations for which an automaton $\mathcal{A}$ should be identified. An automaton learning algorithm identifies $\mathcal{A}$ weakly polynomially in the limit with probability one if $\mathcal{A}$ is found based on observations $\mathcal{O}, |\mathcal{O}| > n$ for some $n \in \mathbf{N}$ and if the algorithms run in $O(f(|\mathcal{O}|)$ where $|\mathcal{O}|$ denotes the length of all observations and f a polynomial function.*

Lemma 3. *The algorithm HyBUTLA (Algorithm 1) runs in $O(m^3)$ where m denotes the number of nodes in the original prefix tree.*

Proof: *Step (1) of the algorithm runs in $O(m)$ since the number of events is less than the number of observations. Step (2) also uses every observation once, i.e. it runs in $O(m)$. The algorithm HyBUTLA from Algorithm 1 compares in step (3) a maximum of m^2 nodes. In each merging step for nodes v, w, their both subtrees have to be accessed by the algorithm compatible and determinized.* □

Theorem 3. *If $\forall i,j : p_i - p_j > 2\gamma$, the algorithm HyBUTLA fulfills the criterion of "Weak Polynomial Identification in the Limit with Probability One".*

Proof. Follows from corollary 2 and from lemma 3. □

Definition 4. *(Strong Polynomial Identification in the Limit with Probability One) Let $\mathcal{O}$ be a set of observations for which an automaton $\mathcal{A}$ should be identified. An automaton learning algorithm identifies $\mathcal{A}$ strongly polynomially in the limit with probability one if $\mathcal{A}$ is found*

(a) with a probability of $> (1-\delta), \delta \in \mathbf{R}$ based on observations $\mathcal{O}, |\mathcal{O}| > n$ for some $n \in \mathbf{N}$ and

(b) if $n = f_2(|\mathcal{A}|, \frac{1}{\delta})$ and

(c) if the algorithms run in $O(f_1(|\mathcal{O}|, \frac{1}{\delta})$ where $|\mathcal{O}|$ denotes the length of all observations and f_1, f_2 are polynomial functions.

Theorem 4. *If $\forall i, j : p_i - p_j > 2\gamma$ and if the automaton size $|\mathcal{A}|$ is larger than $|\Sigma|$, the algorithm HyBUTLA fulfills the criterion of "Strong Polynomial Identification in the Limit with Probability One".*

Proof. We already know from theorem 3, that "Weak Polynomial Identification in the Limit with Probability One" is fulfilled.

The error probability of HyBUTLA is bounded by $c_1 \alpha n^3 |\Sigma|, c_1 \in \mathbf{R}$ (see theorems 1 and 2). So to achieve an error probability of $< \delta$, we set $\alpha = \frac{c_2}{n^4}, c_2 \in \mathbf{R}$. This fulfills point (a) from above.

From this follows also that the criteria are fulfilled for $n \geq \frac{c_3|\Sigma|}{\delta}$. And since we assume that the automaton size $|\mathcal{A}| > |\Sigma|$[3], it follows that the criteria are also fulffiled for $n > \frac{c_3|\mathcal{A}|}{\delta}$. From this follow points (b) and (c) from above. □

Probably Approximately Correct Learning

Probably approximately correct learning (PAC learning) is a well-known framework for analyzing learning algorithms:

Definition 5 (PAC Learning)**.** *Let X denote the set of samples (observations), $\mathcal{C}$ a to-be-learned concept class (i.e. $C \in \mathcal{C} \Rightarrow C \subseteq X$) and D a probability distribution over X. The function χ_C is the classification whether a sample belongs to a concept $C \in \mathcal{C}$, i.e. $\chi_C : X \rightarrow \{0, 1\}$ and $\forall C \in \mathcal{C}, x \in X : \chi_C(x) = 1 \Leftrightarrow x \in C$.*

$\mathcal{C}$ is called PAC-learnable iff for all concepts $C \in \mathcal{C}$, for all distributions D, for all $0 \leq \epsilon < 0.5$ and for $0 \leq \delta < 0.5$ an algorithm A exists which draws $n \in \mathbf{N}$ samples according to D and for which the following holds:

(i) A's result, i.e. a hypothesis $H \in \mathcal{C}$, has an error of $\leq \epsilon$,

(ii) A finds this hypothesis H with a probability $\geq (1-\delta)$,

(iii) A runs polynomial in n,

(iv) A runs polynomial in $\frac{1}{\epsilon}$,

(v) A runs polynomial in $\frac{1}{\delta}$,

Lemma 4. *The learning error ϵ of HyBUTLA is bounded by $O(m^2|\Sigma|\sqrt{\frac{1}{n}\log\left(\frac{2}{\alpha}\right)})$ where n denotes the number of input samples and m the number of states of the generating automaton, i.e. of the correct automaton.*

[3]Which only means that there are no unused symbols in the alphabet.

Proof. So at least one of $\frac{m(m-1)}{2}$ compatibility checks have failed where each such check comprises $(2|\Sigma| + 1)$ tests of the Hoeffding bounding. Presuming again that g_0 and g_1 from the Hoeffding test (equation (1)) grow linearly with the number of observations ($g_0 = c_0 n, g_1 = c_1 n$), the term

$$\epsilon < \frac{m(m-1)}{2} \cdot (|\Sigma| + 1) \cdot \sqrt{\frac{1}{2}\log\left(\frac{2}{\alpha}\right)} \left(\frac{1}{\sqrt{g_o}} + \frac{1}{\sqrt{g_1}}\right) = O(m^2|\Sigma|\sqrt{\frac{1}{n}\log\left(\frac{2}{\alpha}\right)})$$

bounds the error. □

Theorem 5. *The concept class $\mathcal{C}_\mathcal{H}$ of hybrid automata according to definition 1 is PAC-learnable under two constraints:*

- *$\forall i, j : p_i - p_j > 2\gamma$ (probabilities appear only in step sizes) and*
- *all observations $\frac{f_i}{g_i}\ g_i$ are proportional to n (the number of times a transition is triggered is proportional to the number of samples)*

Proof. The proof shows that $\mathcal{C}_\mathcal{H}$ is PAC-Learnable by HyBUTLA. For this, all single features of the definition 5 will addressed. Most parts of the proof will relate to theorem 4, please note that this proof makes no assumptions about the distribution or concepts except the two mentioned above.

(i) Lemma 4 bounds the error.

(ii) Follows directly from theorem 4.

(iii) Follows directly from theorem 4.

(iv) Follows from lemma 4, since m and ϵ have a polynomial relation and HyBUTLA run according to lemma 3 in polynomial time.

(v) Follows directly from theorem 4.

From lemma 4 follows that HyBUTLA has a maximum error of $O(m^2|\Sigma|\sqrt{\frac{1}{n}\log\left(\frac{2}{\alpha}\right)})$ where n denotes the number of input samples and m the number of states of the generating automaton. □

5.2 Errors with Incorrect Continuous Signal Values

Incorrectly learned models for the continuous signal values can be attributed to two error sources:

1. Inadequate description of the data with random walk models.
2. Insufficient amount of data for parameter estimation.

It is not possible to provide a general error estimate for the first error source because the modeling error depends on the concrete physical process described with the model. For example, sub-processes which can be executed in an arbitrary order cannot sufficiently described with a single model sequence (see [Windmann *et al.*, 2013]). Further, the assumption of normally distributed state and measurement noise in the process models does not hold for every process.

The second error source can be analyzed under the assumption of negligible modeling errors. As the model parameters in eq. (1) are obtained with maximum likelihood estimation, the estimates are converging in the limit to the

optimal values [Ghahramani and Hinton, 1996]. The errors for the states S of the automaton can be obtained as

$$Var(\theta) = -\frac{1}{\frac{\partial^2}{\partial\theta^2}E[lnL(\theta|y(0)\dots y(N))]}, \quad (3)$$

where N denotes the number of samples for learning each model in the sequence and $E[\dots]$ the expectation value of the log-likelihood $lnL()$ which is maximized according to [Ghahramani and Hinton, 1996]. Evaluation of eq. (3) leads to error estimates

$$Var(\sigma_s^{(u)}) = \frac{{\sigma_s^{(u)}}^2}{2N}, Var(\sigma_s^{(v)}) = \frac{{\sigma_s^{(v)}}^2}{2N}. \quad (4)$$

I.e., the estimation errors for the model parameters decrease linearly with the number of learning samples and are thus neglected in the further considerations.

The assessment of incorrectly classified measurements under the assumption of insignificant modeling errors and correctly learned models is straightforward: As described in [Windmann *et al.*, 2013], measurements $y(k)$ beyond a given confidence interval $1-\beta$ of the distribution $p(y(k)|y(0)\dots y(k-1))$ which is obtained by Kalman filtering are considered as outliers. Thus, the probability for each measurement of being a false positive is just the probability β that the measurement is beyond the confidence interval. As the measurements $y(k), k = 0\dots K$, can be considered to be randomly drawn from the distribution $p(y(k)|y(0)\dots y(k-1))$, the probability P_{FP} of a false-positive sequence of K measurements is obtained as

$$P_{FP} = 1-(1-\beta)^K. \quad (5)$$

Further details can be found in [Windmann *et al.*, 2013].

6 Applications of the Algorithm

The proposed anomaly detection algorithm was applied on the Lemgo Model Factory. In this production plant, the raw material (corn) is transported and processed (resulting in popcorn). The Lemgo Model Factory comprises several components including: a storage system, a conveyor belt, a robot, and a popping machine.

For learning the behavior model of one of the production modules (the popping machine), we used data recorded in a production cycle with training data. In the experiments, the active power of the popping machine was considered as output of the behavior model. The power range of the popping machine is $0-1150W$. The model predicts this output, while the anomaly detection algorithm detects its discrepancies from the observed signal values (see Figure 1). Figure 5 b) shows the time diagram of the output signal, taken in one of the recorded production examples (with the mean value and the standard deviation $276.97 \pm 16.51W$). For simplicity, we considered as input signals only two discrete signals, which indicate whether heater and fan are active. In this approach, an underlying finite state machine with four system modes is assumed which comprises the four combinations of these two signals (see Figure 5 a). Events, i.e. changes of these two signals, lead to mode transitions.

Once the model was learned, ANODA was executed on additional test cycles. The anomaly detection was performed targeting the following faults: Zero value of the signal (f1), signal drops (f2) and signal jumps (f3). The faults f1, f2 and f3 were detected when the process (continuous)

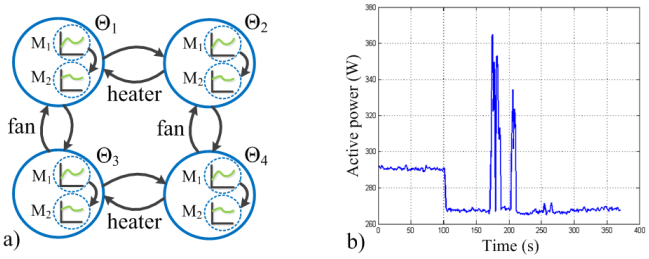


Figure 5: Popping machine: a) Behavior model b) Energy consumption.

output variable was out of range. In order to evaluate the results, as in [Vodenčarević *et al.*, 2011] the Sensitivity, Specificity, Accuracy and F measure are used (all expressed as a percentage).

- Sensitivity (true positive rate): TP/(TP + FN)).
- Specificity (true negative rate): TN/(TN + FP)).
- Accuracy: ((TP + TN)/(TP + TN + FP + FN)).
- F measure: $2 \cdot ((\text{Sensitivity} \cdot \text{Precision})/(\text{Sensitivity} + \text{Precision})$ with Precision = TP / (TP + FP).

Table 2 shows the average sensitivity, specificity and accuracy calculated over 100 runs that were performed for each of the three faults. Sensitivity, specificity and accuracy rates of 100% were obtained for signal drops and jumps by 10% ($f_1^{(10\%)}, \dots, f_3^{(10\%)}$). Even signal drops and jumps of 2.5% led to the same results.

Table 2: Performance metrics for the three faults

Fault	Sensitivity	Specificity	Accuracy	F meas
f_1	100%	100%	100%	100%
$f_2^{(10\%)}$	100%	100%	100%	100%
$f_2^{(2.5\%)}$	100%	100%	100%	100%
$f_2^{(1.5\%)}$	83.33%	100%	87.50%	90.91%
$f_2^{(1\%)}$	36.67%	100%	52.50%	53.66%
$f_3^{(10\%)}$	100%	100%	100%	100%
$f_3^{(2.5\%)}$	100%	100%	100%	100%
$f_3^{(1.5\%)}$	66.67%	100%	75%	80.00%
$f_3^{(1\%)}$	20%	100%	40%	33.34%

These results show that new algorithms HyBUTLA and ANODA work well on real-word data.

7 Conclusion and Outlook

In this paper, the capabilities of the ANODA algorithm are analyzed. This algorithm uses hybrid timed automata behavior models for the anomaly detection. These models are learned using the HyBUTLA algorithm with the incorporated linear state space models for describing the continuous dynamics within the automaton states. It has been formally shown that errors of the event sequences disappear when the model is learned in the limit, i.e. from sufficient amount of data. Errors in learned timing constraints and continuous behavior can be kept at small values. Conducted experimental results verify the usability of combined model-learning and anomaly detection approaches in the real-world.

In future work, various algorithms for learning the continuous behavior in automaton states will be applied and case studies will be conducted in various technical systems.

References

[Alur and Dill, 1994] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, vol. 126:183–235, 1994.

[Alur *et al.*, 1995] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[Carrasco and Oncina, 1994] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *GRAMMATICAL INFERENCE AND APPLICATIONS*, pages 139–152. Springer-Verlag, 1994.

[Carrasco and Oncina, 1999] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. In *RAIRO (Theoretical Informatics and Applications)*, volume 33, pages 1–20, 1999.

[David and Alla, 2001] R. David and H. Alla. On hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1-2):9–40, January 2001.

[de la Higuera, 1997] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, May 1997.

[Ghahramani and Hinton, 1996] Z. Ghahramani and G. E. Hinton. Parameter estimation for linear dynamical systems. Technical report, Univ. Toronto, Dept. Comput. Sci., 1996.

[Gold, 1967] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gold, 1978] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[Henzinger *et al.*, 1998] Thomas A. Henzinger, Peter W. Kopkeb, Anuj Puria, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1), August 1998.

[Henzinger, 1996] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–292, Washington, DC, USA, 1996. IEEE Computer Society.

[Higuera and Thollard, 2000] Colin Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In ArlindoL. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 141–156. Springer Berlin Heidelberg, 2000.

[Narasimhan and Biswas, 2007] S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 37(3):348 –361, May 2007.

[Niggemann *et al.*, 2012] Oliver Niggemann, Benno Stein, Asmir Vodenčarević, Alexander Maier, and Hans Kleine Büning. Learning behavior models for hybrid timed systems. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, pages 1083–1090, Toronto, Ontario, Canada, 2012.

[Oncina and Garcia, 1992] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Machine Perception and Artificial Intelligence*. World Scientific,, 1992.

[Pethig and Niggemann, 2012] Florian Pethig and Oliver Niggemann. A process data acquisition architecture for distributed industrial networks. In *Embedded World Conference*, 2012.

[Thollard *et al.*, 2000] Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. of the 17th International Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, 2000.

[Verwer *et al.*, 2008] S. Verwer, M. de Weerdt, and C. Witteveen. Polynomial distinguishability of timed automata. In *Proc. of the 9th intl. colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '08, pages 238–251, Berlin, Heidelberg, 2008. Springer-Verlag.

[Verwer *et al.*, 2009] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. One-clock deterministic timed automata are efficiently identifiable in the limit. In Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martin-Vide, editors, *Language and Automata Theory and Applications*, pages 740–751. Springer, 2009.

[Verwer, 2010] Sicco Verwer. *Efficient Identification of Timed Automata: Theory and Practice*. PhD thesis, Delft University of Technology, 2010.

[Vodenčarević *et al.*, 2011] Asmir Vodenčarević, Hans Kleine Büning, Oliver Niggemann, and Alexander Maier. Using behavior models for anomaly detection in hybrid systems. In *Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on*, pages 1–8, 2011.

[Windmann *et al.*, 2013] Stefan Windmann, Shuo Jiao, Oliver Niggemann, and Holger Borcherding. A stochastic method for the detection of anomalous energy consumption in hybrid industrial systems. In *INDIN*, 2013.

Comparison for Sensor placement algorithms

Yi Dong and Gautam Biswas
Dept. of EECS/ISIS
Vanderbilt University
Nashville, TN 37235. USA

Abstract

Sensor placement algorithms find a minimum set of measurements that can provide maximum detectability and isolability for a set of faults that can occur in the system. Adding sensors to the system usually incurs additional cost and may reduce system reliability (because sensors themselves may degrade and fail). Therefore, sensor placement algorithms play an important role in system design and deployment. In this paper, we study two different algorithms and prove their equivalence. We also compare their computational complexities.

Keywords: *Dulmage-Mendelsohn Decomposition, A* algorithm, Detectability, Isolability, sensor selection, computational complexity*

1 Introduction

An important aspect of designing effective diagnosers involves determining the right set of sensor locations for detection and isolation of known faults in the system [1][13]. Sensor placement algorithms attempt to find a minimum number of sensors that can detect and isolate a known set of faults in a system given a system model [7][11]. Additional criteria, such as likelihood of fault occurrence and sensor failures [15], cost of sensor placement [2], and reliability of the overall diagnostic system [12] have also imposed in determining the minimum sensor set.

Sensor placement algorithms are also sensitive to the nature of the model employed. In most cases, the sensor placement algorithms are based on structural models of the system. This includes graph-based (e.g., [1][11]) and equation-based (e.g., [3][7][13]) models. The work described in this paper focuses on two algorithms to determine optimal sensor placement: (1) a systematic sensor placement algorithm [7] based on the Dulmage-Mendelsohn (DM) structural Decomposition of an equation-based model of a dynamic system [4]; and (2) an A*-based search algorithm from AI that uses a graph-based model of the system dynamics [9] and [11]. Furthermore, [9] much like the methods presented in [1] use signed-directed graphs for deriving fault signatures to represent the effects of fault parameters on observations or sensor measurements.

Our intent in this paper is not to compare different modeling approaches, and their impact on sensor placement algorithms. Instead, we adopt the equation-based framework [3][7][13] and compare the effectiveness of the algorithm based on DM decomposition in [7] with the heuristic A*-based search algorithm in [11]. The original A*-based algorithm used fault signatures derived from a Temporal Causal graph model of the system [9][11]. However, when comparing the two algorithms, we use a Fault Detectability Matrix (FDM) as the starting point for sensor placement using the A* algorithm to achieve a level playing field. The FDM matrix does not use the sign of the change associated with a measurement residual when computing the detectability of a fault given a measurement. We evaluate the ability of these algorithms to generate optimal sensor placement, and also compare their computational complexities.

Sensor placement is implemented offline during system design, taking into consideration factors such as observability, monitorability, controllability, and reliability of the system. Therefore, computational complexity may play an important factor in determining the smallest number of sensors required for full diagnosability, while also meeting some of the other requirements of the designed system.

2 Background

Structural diagnosis methods fall into two categories: graph-based and equation-based methods. However, the task of optimal sensor placement and the notions of detectability and diagnosability can be formally defined independent of the two representations. We begin with the definitions and then present details of the two algorithms under study in the rest of this section.

2.1 Definitions

Given a dynamic system described by its corresponding model, M, a set of faults, F, in this system that are of interest, and a possible set of sensor locations, S, we define the important notions of fault detectability and isolability, and use these definitions to formalize the sensor placement problem. We assume that the occurrence of a fault $f \in F$ causes one or more measurements, $m \in M$ to deviate from its nominal value, producing non-zero residuals that can be measured.

Definition (Fault Detectability) *Given a model M, associated faults F, and possible sensor locations S, iff for all faults $f_i \in F$, we can generate at least one measurement residual which is sensitive to f_i, we say the fault set F is detectable by sensor set S.*

Definition (Fault Isolability) *Given a model M, associated faults F, possible sensor locations S, iff for all pairs of faults $\{f_i, f_j \mid f_i, f_j \in F; f_i \neq f_j\}$, we can generate at least one measurement residual where the residual generated by f_j is different from the residual generated by f_i, we say the fault set F is isolable by sensor set S.*

For a system model *M*, given a set of faults, *F*, and a set of measurements, *S*, it is clear that detectability is a pre-requisite for isolability. Our focus is on isolability, therefore, the next step is to find a minimal set of sensors that make all the faults in *F* isolable. To do this we adapt the definition of minimal sensor set [7].

Definition (Minimal Sensor Set)*: Let S be the set of possible sensor locations, and let $\mathscr{S}$ be a multiset defined on S. Then, $\mathscr{S}$ is a minimal sensor set, if the sensors in $\mathscr{S}$ fulfill the detectability and isolability specification and all proper subsets of $\mathscr{S}$ do not.*

The notion of multiset is used because a sensor may be used more than once to isolate faults. In the next two sections, we briefly review the two sensor placement algorithms as they are originally defined.

2.2 Sensor placement using DM-Decomposition

DM decomposition [4] assumes that the system model is represented by a bipartite graph, with the set of equations represented as one type of nodes and the set of variables as a second type of nodes. The set of equations are rows in the structural model while variables representing system behavior appear in the columns. Fig. 1 illustrates a structural model from [7]. The gray areas imply edges between equations and the variables. By appropriate permutation of rows and columns, the structural model can be represented in an upper block triangular form, i.e., the Dulmage-Mendelsohn (DM) Decomposition. The method to derive the upper block triangular form is described in [4].

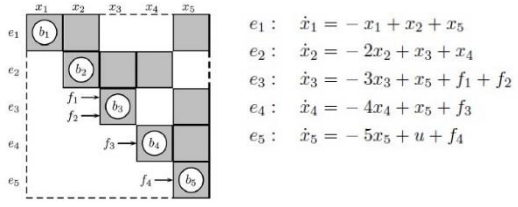


Figure 1. From [7]. Structural model representing the system described by the set of five first order differential equations on the right side. *e* stands for equation, *x* are unknown variables, *f* are faults, and *b* are strongly connected component blocks.

Starting with the set of equations in Fig. 1 the DM-Decomposition partitions the structural model into three parts: (1) *structurally underdetermined* part, which has more variables than equations; (2) *structurally just-determined* part, which has equal number of equations and variables; and (3) *structurally over determined* part, which has more equations than variables. The decomposition can be derived from the bipartite graph by finding the maximum matching, which is actually the non-zero diagonal in the structural model [4]. By appropriate permutation, we can then derive the block triangular form of the matrix [12], and this provides the three structural parts. The over determined part of the system equations are redundant, and form the basis for the fault diagnosis algorithm. The example shown in Fig. 1 is a just-determined system with the upper block triangular form that has a nonzero diagonal. This defines a perfect matching in the bipartite graph [5]. Adding sensors introduces additional equations, which become the over-determined part of the system model. The over-determined part makes fault parameters, $f \in F$, detectable.

Deriving sets of sensors that achieve maximum isolability is a special case of maximum detectability This is achieved by removing the equation which makes fault f_i detectable; if all other faults are still detectable, then fault f_i is isolable for the set of chosen sensors. Therefore, by selectively adding appropriate sensors as we cycle through all faults in *F*, the set *F* can be made isolable.

In order to be efficient, the algorithm applies the notion of Hasse diagram of the partial order over blocks and faults classes [2]. The Hasse diagram is derived according to orientations of unknown variables and equations [3], which is defined as follows: (a) if the edge (e_i, x_j) belongs to the matching (on the non-zero diagonal), it is oriented from e_i to x_j; and (b) otherwise, it is oriented from x_j to e_i.. Faults associated with topmost blocks (maximum faults classes) in the Hasse diagram are blocks that cannot be oriented to other faults-associated blocks. Sensor sets that can detect faults in maximum fault classes can detect all other faults because the fault classes associated with all other blocks have propagation paths to maximum fault class blocks according to the orientation. And since a block is strongly connected, any measurement in a block is sufficient to detect all faults within that block. This is shown in Figure 2.

The sensor placement algorithm will first find sensor sets S_D that achieve maximum detectability and then repeatedly remove the equation associated with a fault to find sensor sets S_I achieving maximum detectability for the rest of the system. By applying the Minimal Hitting Set algorithm [7] on resulting sensor sets S_D and S_I, we can get S, the sensor set that has a non-empty intersection with all of them, and it maximizes detectability and isolability.

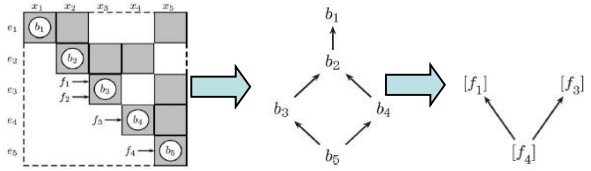


Figure 2. Procedure of generating Hasse diagram of partial order over blocks (right) and faults classes (bottom), where $[f_1]$ and $[f_3]$ are maximum faults classes. (This figure is from [7])

2.2 A* based Sensor Placement algorithm

Before describing this algorithm [11], we introduce the notion of temporal causal graph (TCG) and how to derive fault signatures from the TCG. A TCG is a directed graph derived from a bond graph model using causal assignments to the bonds derived from the SCAP algorithm [5], and creating a directed graph that captures all of these relations [9]. A fault signature defines the changes to a sensor value induced by a fault (typically, we define a fault by a $\pm$ change to the parameter value associated with the fault). Propagation starts with 0-order change through all causal paths in the TCG from the fault towards the measurements. Every integrating edge increases the temporal order of the signature by 1 (indicating a delay in the propagation). Since the TCG is a global model, i.e., it contains all relations between variables and faults, it is sufficient to determine fault signatures for all faults on all possible observations in the system model, based on a specific causal assignment. Figure 3 shows this procedure for a simple one tank system. The fault signature

of C on e_2 is +,–, where $C^- \rightarrow e_2^+ \rightarrow e_3^+ \rightarrow f_3^+ \rightarrow f_2^-$, and $f_2 \rightarrow e_2^{'-}$ is a first-order effect because of the integrating edge. This implies an instantaneous jump in e_2 (0^{th} order change) if C decreases, and then a gradual decrease in its residual (1^{st} order change).

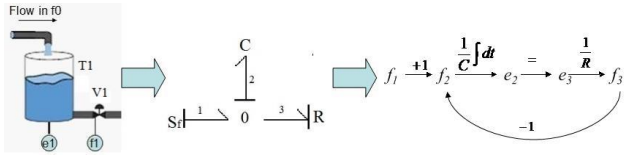


Figure 3. (left) One tank system; (middle) Bond Graph; and (right) Temporal Causal Graph(TCG)

The A* search algorithm finds the optimal path from a start state (no sensors selected) to a goal state (a minimal sensor set). A* introduces two measures: *g*, the distance from the initial state to the current state (i.e., the number of sensors added), and *h*, the estimate of the distance from the current state to the goal state (i.e., estimate of minimum number of sensors needed to achieve full diagnosability) [11]. The algorithm finds the optimal, i.e., lowest-cost path if *h* underestimates the true distance to the goal.

The algorithm is best described by a partitioning approach. The system configuration in each step is {*P*, *OSET*}, where *P* is the partition of faults not yet diagnosed, and *OSET* is the set of sensors that have been selected. The initial configuration is $\{P_{init}, OSET_{init}\}$, where P_{init} is made up of a set that contains all faults (no fault can be isolated), and $OSET_{init}$ is empty. Similarly, the goal state $\{P_{goal}, OSET_{goal}\}$, has each element of P_{goal} with exactly one fault, and $OSET_{goal}$ includes the smallest number of sensors that can isolate all faults. This set may not be unique. To achieve the best sensor placement, at each iteration we add the sensor that has the highest discriminatory power. Fault signatures derived for each fault and possible measurement using the Qualitative Fault Signature scheme [9]. As proven in [9][11] a fault can have one of 4 different signatures on an observation: '+,+', '+,–', '–,–' and '–,+' if magnitude and first order computations are formed for each measurement. Therefore, each observation can discriminate at most 4 sets of faults. This provides the information for deriving *h*. There exist two different way of specifying *h*, and because configurations differ from each other, the two derived values for *h* are not necessarily to be the same. The value h_1 is derived by achieving max partitioning for all partitions. Suppose the current configuration is $\{P_{current}, OSET_{current}\}$, where $P_{current}$ has *t* partitions. A new observation can split each partition into at most 4 new partitions. So $4^h \times t \geq n$, and $h_1 = \log_4(\frac{n}{t})$. The value h_2 is derived by achieving max partitioning of the largest partition. Say, $P_{current}$ has a largest partition P_l that includes t_l faults. It can be split to at most 4 partitions. In order to split it into partitions contain exactly one fault with minimum observations, it should be split into 4 partitions at each iteration. So h_2 can be calculated by $t_l / 4^h = 1$, where $h_2 = \log_4(t_l)$. Then h = Max{h_1, h_2}.

At each iteration *i*, we choose an observation that has lowest value for $h_i + g_i$ to guarantee the underestimate distance from current point to goal point to get an optimal solution on measurement selection. The correctness of the solution has been demonstrated in [11].

The A* algorithm is general, and it can be used with any discriminatory function. In the Case Study we include only binary information, i.e., a sensor measurement can or cannot detect a fault. In this case, a fault set can be decomposed into two sets by a measurement, therefore, the base of the logarithm used for the computation is 2.

3 Case Study

We will use a traditional three tank system configuration shown in Fig. 4 to illustrate the two algorithms. The bond graph model (Fig. 5) implies six sensor variables in the system, that is pressures in the tanks, *p1*, *p2*, *p3*, and flow rates out of the tanks, *q1*, *q2* and *q3* (*q0*, the input flow rate is known). The sensor locations (circled) are shown in Fig. 4. In the bond graph model, the tanks are modeled as capacitors, and the pipes and valves as resistances. 0-junctions represent common effort locations, and 1-junctions imply current flow locations [5].

Following the constituent relations of the basic elements of the bond graph, we derive a set of equations (1):

$p1 = e1 = e2 = e3$
$p2 = e5 = e6 = e7$
$p3 = e9 = e10 = e11$
$q1 = f3 = f4 = f5$
$q2 = f7 = f8 = f9$
$q3 = f11$

In order to clearly represent causality, we introduce three more variables $\dot{p}_1$, $\dot{p}_2$ and $\dot{p}_3$, which represent the derivatives of variables *p1*, *p2* and *p3*, respectively for the DM-Decomposition method.

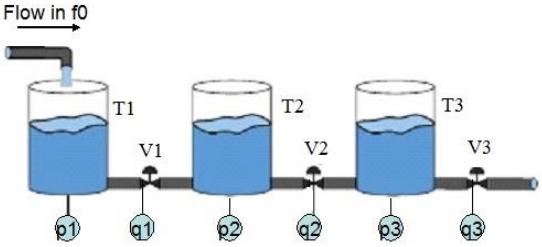


Figure 4. Three-tank system

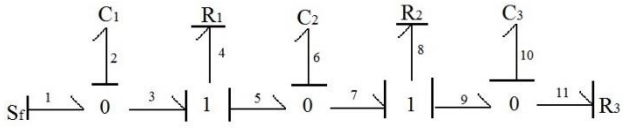


Figure 5. Bond graph model of the three tank system

4 Comparison of the two Methods

4.1 DM-Decomposition method

The set of equations deriving three tanks system behavior is given by (These equations do not imply any causality but only relations between variables and parameters):

E1: $q_1 = \frac{1}{R_1}(p_1 - p_2)$ E2: $q_2 = \frac{1}{R_2}(p_2 - p_3)$

E3: $q_3 = \frac{1}{R_3} \cdot p_3$ E4: $\dot{p}_1 = \frac{1}{C_1}(q_0 - q_1)$

E5: $\dot{p}_2 = \frac{1}{C_2}(q_1 - q_2)$ E6: $\dot{p}_3 = \frac{1}{C_3}(q_2 - q_3)$

E7: $\dot{p}_1 = \frac{dp_1}{dt}$ E8: $\dot{p}_2 = \frac{dp_2}{dt}$

E9: $\dot{p}_3 = \frac{dp_3}{dt}$

The 9 equations and 9 unknowns imply a just-determined system. Six possible single faults associated with the system are R_1, R_2, R_3, C_1, C_2 and $C3$ (faults in the valves and tank capacities, respectively). After proper permutation of equations and unknown variables, we obtain the DM-decomposition shown in Figure 6.

Using the DM-Decomposition method for maximum detectability we get the sensors sets: $\{q_1\}$, $\{q_2\}$, $\{q_3\}$, $\{p_1\}$, $\{p_2\}$, $\{p_3\}$, $\{\dot{p}_1\}$, $\{\dot{p}_2\}$, $\{\dot{p}_3\}$, and for each variable corresponding to a sensor, we add an equation that measures the unknown variable. This makes all faults detectable. Since the system represents a strongly connected component of the structural model, any one sensor makes all faults detectable. And by repeatedly remove one equation associated with a fault we find sensors that achieve full detectability for the rest of the system. The union of the sensor sets after each iteration defines the sensor set that achieves maximum isolability. Among the list of completely diagnosable sensor sets of size 3 are$\{\dot{p}_1, q_1, q_3\}$, $\{p_1, q_1, q_3\}$, $\{p_2, q_1, q_2\}$, $\{\dot{p}_2, q_1, q_2\}$[1].

	q_1	q_2	q_3	p_1	p_2	p_3	$\dot{p}_1$	$\dot{p}_2$	$\dot{p}_3$
E_1	1	0	0	1	1	0	0	0	0
E_2	0	1	0	0	1	1	0	0	0
E_3	0	0	1	0	0	1	0	0	0
E_7	0	0	0	1	0	0	1	0	0
E_8	0	0	0	0	1	0	0	1	0
E_9	0	0	0	0	0	1	0	0	1
E_4	1	0	0	0	0	0	1	0	1
E_5	1	1	0	0	0	0	0	1	0
E_6	0	1	1	0	1	0	0	0	1

Figure 6. DM-decomposition of the example system, where the whole system is in one strongly connected component. Notice there is a nonzero diagonal in the matrix which is a perfect matching of the bipartite graph from the system.

4.2 A* approach

For comparison purposes, we use the same diagnosability information for the A* algorithm as the DM-Decomposition. This information is introduced as the Fault Detection Matrix (FDM). So this is a modified version A* algorithm. The FDM for the three tank system is shown in Figure 7. In this matrix, a "1" for $\{s_i, F_i\}$ implies that if F_i is detectable when removing all other faults from the system under s_i, which actually means, F_i has a propagation path to s_i. For example, R_1 is detectable under measurement q_1 when removing equations E_2, E_3, E_4, E_5, E_6 associated with R_1, R_2, C_1, C_2, C_3, respectively(propagation path: $R_1 \rightarrow E_1 \rightarrow q_1$).

$m \backslash F$	C_1	C_2	C_3	R_1	R_2	R_3
q_1	1	1	0	1	0	0
q_2	0	1	1	0	1	0
q_3	0	0	1	0	0	1
p_1	1	0	0	1	0	0
p_2	0	1	0	1	1	0
p_3	0	0	1	0	1	1
$\dot{p}_1$	1	0	0	1	0	0
$\dot{p}_2$	0	1	0	1	1	0
$\dot{p}_3$	0	0	1	0	1	1

Figure 7. Fault Detection Matrix
Fault Detection Matrix is a two dimensional matrix that contains detectability information of each pair of $\{O, F\}$, where O is an observation, F is a fault

Initially, $P_{init} = \{C_1, C_2, C_3, R_1, R_2, R_3\}$, and all observations can split the partition. Calculating, $h = \text{Max}\{h_1, h_2\}$, we get

$$h_{q_1} = \max\{\log_2 3, \log_2 3\} = \log_2 3$$
$$h_{q_2} = \max\{\log_2 3, \log_2 3\} = \log_2 3$$
$$h_{q_3} = \max\{\log_2 3, \log_2 4\} = \log_2 4$$
$$h_{p_1} = \max\{\log_2 3, \log_2 4\} = \log_2 4$$
$$h_{p_2} = \max\{\log_2 2, \log_2 3\} = \log_2 3$$
$$h_{p_3} = \max\{\log_2 3, \log_2 3\} = \log_2 3$$
$$h_{\dot{p}_1} = \max\{\log_2 3, \log_2 4\} = \log_2 4$$
$$h_{\dot{p}_2} = \max\{\log_2 2, \log_2 3\} = \log_2 3$$
$$h_{\dot{p}_3} = \max\{\log_2 3, \log_2 3\} = \log_2 3$$

Since $g = 0$ initially, and there are some tied values for $\min\{h\}$, we randomly choose one, say q_2 and add it to $OSET$, $OSET = \{q_2\}$, and $P = \{C_1, R_1, R_3\}, \{C_2, C_3, R_2\}$. In the next iteration, we repeat this procedure to re-calculate h values. The minimum value occurs for sensor q_1. Then $OSET = \{q_2, q_1\}$, and $p = \{C_1, R_1\}, \{R_3\}, \{C_2\}, \{C_3, R_2\}$.

In the next and last iteration, we choose p_2 as the next observation, then $OSET = \{q_2, q_1, p_2\}$ which splits p into $\{C_1\}, \{R_1\}, \{R_3\}, \{C_2\}, \{C_3\}, \{R_2\}$, and this implies a fully diagnosable system. This $OSET = \{q_2, q_1, p_2\}$ is one of the optimal set of sensors we derived by using DM-Decomposition method. We could derive the other minimum sensor sets by making different choices when ties occur in the choice of sensors.

4.3 Comparison of the two approaches

According to example results shown above, the A* algorithm derives one of the optimal sensor sets derived by DM-Decomposition approach.

[1] We do not list all possible sensor combinations for minimal diagnosis.

4.3.1 Comparison on functionality

Equivalence for Detectability

The A* algorithm does not have an explicit notion for deriving maximum detectability, so we use the same method as the DMD approach.

Equivalence for Isolability

The A* approach can be seen as tracking a binary search tree shown in Figure 8. There is a set of faults for each vertex, and an observation O_i for each non-leaf vertex, which can split the set into two parts, where the set of faults on its left child is detectable under O_i while the set on its right child is not. So it starts from the root with all faults in one set$\{f_1 \dots f_n\}$, adds an observation O_0 splitting the partition into $\{f_1 \dots f_i\}$ and $\{f_{i+1} \dots f_n\}$. Repeat the procedure until we have full isolation, i.e., $\{f_1\}, \{f_2\} \dots \{f_{n-1}\}, \{f_n\}$. In addition, we may need an observation O_n to make sure the system is fully detectable. $S_{A^*} = \{O_0, O_1 \dots O_k \dots O_p\} \cup \{O_n\}$ is an optimal sensor set for maximum detectability and isolability.

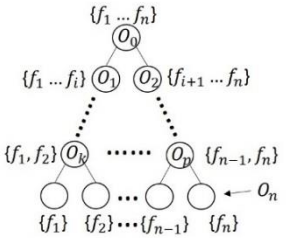


Figure 8. Binary Search Tree for A* algorithm

For DM-Decomposition, we start from the leaves of the binary search tree, removing equations associated with a fault one by one and getting sensor sets that detect the rest of the faults. The minimal hitting set provides the optimal sensor placement.

So, we want to prove that S_{A^*} is also an optimal answer derived by DMD method. In order to do this, we need to prove that there is always a set $S_i \subseteq S_{A^*}$, which is sufficient to detect all faults in the system $M_s \backslash e_{f_i}$.

Proof: Notice that when a vertex in the binary tree is a left-child of its parent, all faults in that partition are detectable under its parent's observation. So all faults of its descendants are also detectable. Obviously, every leaf that is a left child can be detectable under its parent's observation, i.e., f_1 is detectable under O_k. And for the leaf which is a right child, we just need to climb the tree to a level where it reaches a vertex that is a left child of its parent, then the observation of this parent can detect this fault. The only exception is right-most leaf which can only be detected by O_n. So the observation set $S_i \subseteq S_{A^*}$ is always sufficient to detect rest of faults when we remove one from the system. ■

Second, we need to prove that every observation $O_i \in S_{A^*}$ is needed at least once in the iterations of the DMD method if we want only observations from S_{A^*}.

Proof: Notice that the observation of an vertex V_i in binary tree is the only one in S_{A^*} which can detect the right-most leaf-fault of its left sub-tree (every non-leaf vertex can have two sub-tree, left and right) because the right-most leaf of that sub-tree cannot get to an vertex which is a left child by climbing up the tree. And since the root of the sub-tree is the left child of V_i, observation of V_i is needed to detect that right-most leaf-fault. ■

According to the proof above, we can always find an observation set $S_i \subseteq S_{A^*}$ when we run iterations of DMD method, and every $O_i \in S_{A^*}$ will happen at least once in S_i. So after getting the minimal hitting sets, the set S_{A^*} must be one of the resulting sets by DMD method since A* algorithm is guaranteed to give an optimal answer.

To sum up, the A* algorithm and the DMD methods generate equivalent results. The DMD method produces multiple optimal sets, while the A* algorithm only generates one answer. However, if we maintain multiple search trees whenever ties occur in *h*, by making different choices, we get exactly the same results as the DMD method.

4.3.2 Comparison on computational complexity

We compare the computational complexity of the two algorithms for isolability. Let us assume first, the system consists of *n* possible observation locations, which is typically the number of unknown variables, and *m* faults. It is clear that there are at most *k* numbers of iterations in A* algorithm, where $k = |OSET_{final}|$. This is because exactly one observation will be added to *OSET* in each iteration. Then the algorithm will calculate the value of $g+h$ for each observation that does not belong to *OSET* repeatedly, and at most *n* observations will be checked in each iteration. It takes constant time to get the value of g by $g = |OSET_{current}| + 1$. And for *h*, we need to go through all the partitions in $P_{current}$ at least once either to find the largest partition or count the number of partitions. Since there are at most $O(m)$ partitions in each iteration, it take $O(m)$ time to get the value of t_l (size of largest partition)or t (number of partitions), and then obviously, constant time to compute $h_1 = log_4(t)$ and $h_2 = log_4(t_l)$. Overall, the time complexity for the A* algorithm is $k*n*O(m) = O(k*n*m)$, where *k* is the number of observations been selected, *n* is the number of possible sensor locations and *m* is the number of faults.

To perform isolation, the DM-Decomposition algorithm, repeatedly removes one equation associated with a fault class and finds the maximum detectability in the rest of the system. So the algorithm is going to repeat for at most $O(m)$ times. For each iteration, a matching for the bipartite graph is needed to form the non-zero diagonal in decomposition. The computational complexity of finding the maximum matching is $O(E\sqrt{V})$, where *V* is the number of vertices and *E* is the number of edges in the bipartite graph [10]. In our case, there are $2*n$ vertices (*n* unknown variables and typically the same number of equations in the bipartite graph), while the number of edges is at most $O(n^2)$. Overall, this step takes $O(n^{2.5})$ time. Then the step to find strongly connected component can be done in $O(E+V)$ time by using Tarjan's algorithm [14], which is $O(n^2+n)$ in our case. And deriving maximum classes takes the same time as finding strongly connected component by using depth first search. This is an $O(n^{2.5} * m)$ running time.

It is much higher than $O(k*n*m)$ for the A* algorithm. And we have not even take the final step of DMD method: for all sets of observations (derived for all pairs of isolable faults), find the minimum set of observations that has none empty intersection with each of those sets. This problem, called minimum hitting set problem is NP-hard, and it can be deduced from vertex cover problem that is known to be NP-Complete. So the running time would be exponential to the number of pairs of faults. However, if a system is well designed, this part of computation would not take that much time [7]. And since sensor placement analysis is always implementing off-line, the exponential complexity may not be a problem. So the overall computational complexity for the DMD based method is $O(n^{2.5} * m) + U$, where U indicates an unpredictable time value from the NP-hard problem. We are analyzing the worst cases for both algorithms, so the lower bound computations are omitted.

5 Conclusion and discussion

We have shown that the detectability and isolability algorithms for the A* and DMD approaches are equivalent, and we can infer that they have the same discriminatory capability. For observations which can accommodate higher order and multiple types of deviations, we can introduce integral causality into the DM-Decomposition as well as making different observation type. Different observation types can have multiple measurement equations on the same measurement location. By following the DMD procedure, we can find the maximum detectability and isolability using fault signatures. The two algorithms are equivalent in functionality.

DM-Decomposition based methods have a complexity given by $O(n^{2.5} * m)+U$, whereas it is $O(k*n*m)$ for the A* algorithm. One of the reasons for this is that the DM-Decomposition based method repeats some iterations unnecessarily, i.e., doing exhaustive search for all possible fault pairs. For example, when both f_i and f_j are detectable, if f_i is isolable from f_j, it is not necessarily true that f_j is also isolable from f_i by the notion of faults being detectable in the overdetermined part of a system. However, this is actually true, because f_i will have deviation on at least two observations while f_j has deviation on only one of them as we have discussed above, both of them are isolable from each other. On the other hand, the DM-Decomposition based algorithm can give us all possible sets of sensors locations that can achieve maximum isolability, while A* algorithm can only derive them sequentially, when there are ties in the $g+h$ value among some observations. Besides, the accuracy or timeliness of diagnosis can decrease as the number of sensors is decreased, because it implies longer propagation paths from faults to the sensors. The accuracy or timeliness of detection and isolation may decrease becomes significant for higher order systems. Therefore, if we consider accuracy and timeliness of diagnosis, the DM Decomposition based algorithm may provide better results because one can choose additional criteria to pick the sensor set that best satisfies these criteria.

References

[1] M. Bhushan and R. Rengaswamy, "Design of sensor location based on various fault diagnostic observability and reliability criteria", *Computers and Chemical Engineering*, 24(2): 735-741, 2000.

[2] R. Duan, O. Dongxiu, D. Decun, Z. Huilin, "Optimal Sensor Placement for Fault Diagnosis Based on Diagnosis Cost Specifications" *Journal of Computational Information Systems*, 7(9): 3253-3260, 2011.

[3] V. de Flaugergues , V. Cocquempot, M. Bayart and M. Pengov, "Structural Analysis for FDI: a modified, invertibility-based canonical decomposition", *Proc. 20th Intl Workshop on Principles of Diagnosis (DX-09)*, Stockholm, Sweden, 59-66, 2009.

[4] A. L. Dulmage and N. S. Mendelsohn, "Coverings of bipartite graphs," Can. J. Math., 10(4): 516–534, 1958.

[5] Karnopp, D. C., Rosenberg, R. C. and Margolis, D. L., *System dynamics: a unified approach*", Wiley, ISBN 0-471-62171-4, 1990.

[6] E. Frisk, A. Bregon, J. Aslund, M. Krysander, B. Pulido and G. Biswas, "Diagnosability Analysis Considering Causal Interpretations for Differential Constraints", IEEE Transactions on Systems, Man and Cybernetics. Part A, Systems and humans, 42(5), 1215-1229, 2012.

[7] M. Krysander and E. Frisk, "Sensor Placement for Fault Diagnosis, *IEEE Trans on Systems, Man, and Cybernetics—Part A: ystems and Humans*, 38(6): 1398-1410, 2008.

[8] D. Maquin, M. Luong, and J. Ragot. "Fault detection and isolation and sensor network design," *European Journal of Automation*, 31(2): 393–406, 1997.

[9] Mosterman, P., and G. Biswas, "Diagnosis of Continuous Valued Systems in Transient Operating Regions", *IEEE Transactions on Systems, Man, and Cybernetics*, 29(6): 554-565, 1999.

[10] Micali, S. and Vazirani, V. V., "An scriptstyle $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs," *Proc. 21st IEEE Symp. Foundations of Computer Science*, 17–27, 1980.

[11] S. Narasimhan, P.J. Mosterman and G. Biswas, "A systematic analysis of measurement selection algorithms for fault isolation in dynamic systems," *Proc. 9th Intl Workshop on Principles of Diagnosis (DX-98)*, Cape Cod, MA, 94-101, 1998.

[12] A. Pothen and C.J. Fan, "Computing the Block Triangular Form of a Sparse Matrix," ACM Transactions on Mathematical Software, 16(4): 303-324, 1990.

[13] A. Rosich, A.A. Yassine, and S. Ploix, "Efficient Optimal Sensor Placement for Structural Model Based Diagnosis," *Proc. 21th Intl Workshop on Principles of Diagnosis (DX-10)*, Portland, OR, 59-66, 2009.

[14] Tarjan, R. E. (1972), "Depth-first search and linear graph algorithms", SIAM Journal on Computing 1 (2): 146–160, doi:10.1137/0201010

[15] B. R. Upadhyaya, S. R. P. Perillo, X. Xu, and F. Li, "Advanced Control Design, Optimal Sensor Placement, and Technology Demonstration for Small and Medium Nuclear Power Reactors*," Proc. 17th Intl Conf on Nuclear Engineering (ICONE17)*, 763-773, 2009.

Exploiting Parse Trees in LTL Specification Diagnosis

Ingo Pill and Thomas Quaritsch
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria,
{ipill, quaritsch}@ist.tugraz.at

Abstract

Specifications are a development process' lifeblood. Capturing the designers' intentions regarding functionality, interface, test targets, and other aspects, they establish the correct context in design communication, development, and verification amongst other steps like synthesis. A specification's quality is thus a crucial factor. Recently we showed a way to exploit model-based diagnosis for the development of formal (functional) specifications in the Linear Temporal Logic (LTL). In this paper we show how to improve that diagnosis process' search via considering a specification's parse tree. Implementing our ideas with the well-established HS-DAG algorithm, we report experimental results showing our reasoning's attractiveness.

1 Introduction

Specifications capturing the design intent are an essential instrument in any design process. As the basic vehicle for establishing the context in design-related communication, specifications may define a system's functionality, interfaces, test goals, and many other aspects. Consequently, they drive a system's creation, testing, verification, and potentially even a synthesis process. While it is thus not unexpected that up to 50 percent of product defects, and up to 80 percent of rework efforts can be traced back to requirement defects [Wiegers, 2001], these figures illustrate the high demand for tools and means that help us in getting our specifications right.

Drawing on the precise syntax and unambiguous semantics of formal functional specifications and their languages, coverage and vacuity concepts emerged [Fisman *et al.*, 2009; Kupferman, 2006] that can help in identifying specification flaws. For an enhanced user experience, formal specification development tools like RAT [Pill *et al.*, 2006] offer workflows that allow a designer to explore and verify a specification's semantics. Complementing such work, we showed recently a way to exploit model-based diagnosis for the development of specifications in the Linear Temporal Logic (LTL) [Pill and Quaritsch, 2013]. Based on behavioral samples (traces) that *unexpectedly* satisfy (witnesses) or contradict (counterexamples) a specification, we can isolate corresponding diagnoses at a specification's operator level regarding the root causes for the encountered issue. As underlying reasoning engine for the verification of diagnostic theories, we use a SAT solver and a corresponding encoding of the specification and the trace.

In the context of weak fault models, we show in this paper how to exploit structural information about a specification (i.e. its parse tree) for speeding up the computation of these diagnoses. That is, inspired by the concept of dominance defined for flow-graphs [Prosser, 1959; Lengauer and Tarjan, 1979] that has been exploited also for digital circuits [Kirkland and Mercer, 1987], we show how to draw on the intuitive observation that "If some subformula can resolve a conflict, this is true also for its parent".

While in the context of our specification models as established in [Pill and Quaritsch, 2013], this offers us no option for design abstraction resulting in smaller models, nor to statically restrict the diagnosis space, we exploit this observation dynamically in the diagnosis algorithm, i.e. the structured search itself. We implemented our reasoning with the well-known HS-DAG algorithm, that is [Greiner *et al.*, 1989]'s version of Reiter's diagnosis algorithm [Reiter, 1987], and report first results regarding the effects in this paper.

Our paper is structured as follows. In Section 2, we cover the preliminaries for our approach as discussed in Section 3. After showing in Section 4 how to implement our reasoning for HS-DAG, we report experimental results in Section 4.1. Section 5 offers a discussion of our approach and future work, as well as corresponding conclusions. Related work is discussed throughout the paper where appropriate.

2 Preliminaries

For our definitions of an infinite trace and the Linear Temporal Logic (LTL) [Pnueli, 1977], we assume a finite set of atomic propositions AP that induces alphabet $\Sigma = 2^{AP}$.

Definition 1. *Let AP be a finite set of atomic propositions, and δ and φ LTL formulae. Then an LTL formula is defined inductively as follows [Pnueli, 1977]:*

- *for any $p \in AP$, p is an LTL formula*
- *$\neg\varphi$, $\varphi \wedge \delta$, $\varphi \vee \delta$, $\mathsf{X}\,\varphi$, and $\varphi \,\mathsf{U}\, \delta$ are LTL formulae*

Definition 2. *A* parse tree *(*syntax tree*) $\mathcal{T}(\varphi) = (V_\varphi, v_\varphi, E_\varphi, l(v \in V_\varphi))$ for an LTL formula φ is a directed, vertex-labeled tree, where*

- *V_φ is the set of vertices such that for each subformula ψ in φ there is exactly one vertex (v_ψ) labeled with ψ,*
- *$l(v)$ is a labeling function for vertices $v \in V_\varphi$ s.t. $l(v_\psi) = \psi$,*

- $v_\varphi \in V$ *is the single root vertex,*
- *and* E *is* $\mathcal{T}$*'s set of edges, s.t. for* $v_{\psi_1}, v_{\psi_2} \in V_\varphi$, $e = (v_{\psi_1}, v_{\psi_2})$ *is in* E*, iff* ψ_2 *is an operand of* ψ_1.

Note that we consider multiple occurrences of a syntactic construct to be multiple subformulae. With $\top$ denoting logic *True/High/1* and $\bot$ denoting logic *False/Low/0*, the popular operators $\delta \,\mathsf{R}\, \sigma$, $\mathsf{F}\,\varphi$, $\mathsf{G}\,\varphi$, and $\delta \,\mathsf{W}\, \sigma$ are syntactic sugar for common formulae $\neg((\neg\delta)\,\mathsf{U}\,(\neg\sigma))$, $\top\,\mathsf{U}\,\varphi$, $\bot\,\mathsf{R}\,\varphi$, and $\delta\,\mathsf{U}\,\sigma \vee \mathsf{G}\,\delta$ respectively. While we showed in [Pill and Quaritsch, 2013] how to encode these operators directly for our LTL SAT encoding (see, e.g., Def. 7), without loss of generality, we focus on the core of LTL in this paper.

LTL is defined in the context of infinite traces, which we define as explicit finite sequences as is usual. Such a finite sequence of length k can describe a single infinite trace only in the form of a lasso-shaped trace (with a cycle looping back from the last step k to $0 \leq l \leq k$). The other option would be to consider the sequence as a prefix, such that it refers to the set of infinite traces that extend it. Note that we always refer to an infinite trace when using the term trace.

Definition 3. *An infinite trace* τ *is an infinite word over letters from some alphabet* Σ *of the form* $\tau = (\tau_0\tau_1 \ldots \tau_{l-1})(\tau_l\tau_{l+1} \ldots \tau_k)^\omega$ *with* $l, k \in \mathbb{N}$, $l \leq k$, $\tau_i \in \Sigma$ *for any* $0 \leq i \leq k$ *and* $(...)^\omega$ *denoting infinite repetition of the corresponding (sub-)sequence. With* τ^i*, we refer to* τ*'s suffix starting with* τ_i.

The LTL core operators' semantics in the context of infinite traces are defined as follows:

Definition 4. *Given a trace* τ *and an LTL formula* φ*,* $\tau(=\tau^0)$ *satisfies* φ*, denoted as* $\tau \models \varphi$*, under the following conditions*

$$
\begin{aligned}
&\tau^i \models p && \text{iff } p \in \tau_i\\
&\tau^i \models \neg\varphi && \text{iff } \tau^i \not\models \varphi\\
&\tau^i \models \delta \wedge \sigma && \text{iff } \tau^i \models \delta \text{ and } \tau^i \models \sigma\\
&\tau^i \models \delta \vee \sigma && \text{iff } \tau^i \models \delta \text{ or } \tau^i \models \sigma\\
&\tau^i \models \mathsf{X}\,\varphi && \text{iff } \tau^{i+1} \models \varphi\\
&\tau^i \models \delta\,\mathsf{U}\,\sigma && \text{iff } \exists j \geq i.\tau^j \models \sigma \text{ and}\\
& && \quad \forall i \leq m < j.\tau^m \models \delta
\end{aligned}
$$

Regarding diagnostic reasoning, we adopt for our presentation the formalizations of Reiter's consistency-oriented theory of diagnosis [Reiter, 1987]. Given a system's set of components *COMP*, assumption predicates $AB(c_i)$ for all $c_i \in$ *COMP* encoding whether c_i behaves *abnormally*, a system description *SD* defining the correct behavior $\neg AB(c_i) \Rightarrow$ *NominalBehavior*(c_i), and some actual observations *OBS* about a system's behavior, the system is considered to be at fault iff *SD*$\cup$*OBS*$\cup\{\neg AB(c_i)|c_i \in$ *COMP*$\}$ is inconsistent. While a minterm in the assumptions defines a specific state of the system, a diagnosis Δ is a subset-minimal set of faulty components that explains the inconsistency and is a subset of at least one "consistent" minterm.

Definition 5. *A* diagnosis *for (SD, COMP, OBS) is a subset-minimal set* $\Delta \subseteq$ *COMP such that SD* $\cup$ *OBS* $\cup$ $\{\neg AB(c_i)|c_i \in$ *COMP* $\setminus \Delta\}$ *is consistent.*

Reiter proposes to compute the set of diagnoses as the minimal hitting sets of the set *CS* of (not necessarily minimal) conflicts for (*SD*, *COMP*, *OBS*).

Definition 6. *A* conflict C *for (SD, COMP, OBS) is a set* $C \subseteq$ *COMP such that SD* $\cup$ *OBS* $\cup$ $\{\neg AB(c_i)|c_i \in C\}$ *is inconsistent. Iff there is no* $C' \subset C$*, such that* C' *is a conflict,* C *is a minimal conflict.*

If not available a priori, Reiter's algorithm is able to derive *CS* on-the-fly, which is an attractive feature in situations where the cardinality of diagnoses is restricted so that only a subset of *CS* has to be computed. For the organization of a structured search, the algorithm creates a node- and edge-labeled tree by iteratively expanding its nodes in breadth-first order, starting with the root node n_0. With $h(n) \subseteq$ *COMP* the set of edge labels on a node n's path from the root ($h(n_0) = \emptyset$), each non-leaf node n is labeled with a conflict C_i s.t. $C_i \cap h(n) = \emptyset$. If not computed a priori, valid C_is are requested from the theorem prover for negative checks whether $h(n)$ could be a diagnosis (i.e. whether *SD* $\cup$ *OBS* $\cup$ $\{\neg AB(c_i)|c_i \in$ *COMP* $\setminus h(n)\}$ is consistent). Affirmative checks (or the absence of such a C in a pre-computed *CS*) define n as leaf (labeled ✓) and then $h(n)$ is considered to be a potential diagnosis. Specific rules pruning the tree ensure that the "final" leafs are indeed (subset-minimal) diagnoses. New nodes are created in a breadth-first manner by constructing for each c_j in a non-leaf node's label C_i an outgoing edge labeled c_j and a corresponding destination node.

When the search for diagnoses is limited to some bound k regarding their cardinality, nodes with $|h(n)| = k$ need not be expanded. [Greiner *et al.*, 1989] corrected Reiter's original formulations that had some minor but serious flaws, and offer an improved algorithm, HS-DAG, that uses a directed acyclic graph (DAG) instead of a tree.

In [Pill and Quaritsch, 2013] we showed a way to exploit model-based diagnosis using weak or strong fault models for the diagnosis of LTL specifications given behavioral samples (traces). In order for the paper to be self-contained, we rehearse the necessary definitions and one theorem from [Pill and Quaritsch, 2013].

Definition 7. *[Pill and Quaritsch, 2013] In the context of a given infinite trace with length* k *and loop-back time-step* l*,* $E_1(\psi)$ *encodes an LTL formula* ψ *using the clauses presented in Table 1, where we instantiate for each subformula* φ *a new variable over time, denoted* φ_i *for time instance* i*. Note that we assume* k *and* l *to be known inside* E_1 *and* R*.*

$$
E_1(\varphi) = \begin{cases} R(\varphi) \wedge E_1(\delta) \wedge E_1(\sigma) & \text{for} \quad \varphi = \delta \circ_1 \sigma \\ R(\varphi) \wedge E_1(\delta) & \text{for} \quad \varphi = \circ_2 \delta \\ R(\varphi) & \text{else} \end{cases}
$$

with $\circ_1 \in \{\wedge, \vee, \mathsf{U}\}$, $\circ_2 \in \{\neg, \mathsf{X}\}$ *and* $R(\varphi)$ *defined as the conjunction of the corresponding clauses in Table 1.*

Definition 8. *[Pill and Quaritsch, 2013] For a given infinite trace* τ *(with given* k*),* $E_2(\tau) = \bigwedge_{0 \leq i \leq k} \left[\bigwedge_{p_i \in \tau_i} p_i \wedge \bigwedge_{p_i \in AP \setminus \tau_i} \neg p_i\right]$ *encodes the signal values as specified by* τ*.*

Theorem 1. *[Pill and Quaritsch, 2013] Assume an updated Table 1, where each clause* c *is extended to* $\overline{op}_\varphi \vee c$*, and an assignment* op *to all assumptions* op_ψ *on* φ*'s various subformulae* ψ*'s correctness. An encoding* $E_{WFM}(\varphi, \tau) = E_1(\varphi) \wedge E_2(\tau)$ *of an LTL formula* φ *and a trace* τ *as of Definitions 7 and 8 is satisfiable,* $SAT(E_{WFM}(\varphi, \tau))$*, iff* $\tau \models \varphi$ *under assumptions* op*.*

φ	Unfolding rationales	I	Clauses	
$\top/\bot$	$\varphi_i \leftrightarrow \top/\bot$	✓	(a)	$\varphi_i/\overline{\varphi}_i$
$\delta \wedge \sigma$	$\varphi_i \leftrightarrow (\delta_i \wedge \sigma_i)$	✓	(b_1)	$\overline{\varphi}_i \vee \delta_i$
		✓	(b_2)	$\overline{\varphi}_i \vee \sigma_i$
		✓	(b_3)	$\varphi_i \vee \overline{\delta}_i \vee \overline{\sigma}_i$
$\delta \vee \sigma$	$\varphi_i \leftrightarrow (\delta_i \vee \sigma_i)$	✓	(c_1)	$\varphi_i \vee \overline{\delta}_i$
		✓	(c_2)	$\varphi_i \vee \overline{\sigma}_i$
		✓	(c_3)	$\overline{\varphi}_i \vee \delta_i \vee \sigma_i$
$\neg\delta$	$\varphi_i \leftrightarrow \neg\delta_i$	✓	(d_1)	$\overline{\varphi}_i \vee \overline{\delta}_i$
		✓	(d_2)	$\varphi_i \vee \delta_i$
$\mathsf{X}\,\delta$	$\varphi_i \leftrightarrow \delta_{i+1}$	✓	(e_1)	$\overline{\varphi}_i \vee \delta_{i+1}$
		✓	(e_2)	$\varphi_i \vee \overline{\delta}_{i+1}$
$\delta\,\mathsf{U}\,\sigma$	$\varphi_i \rightarrow (\sigma_i \vee (\delta_i \wedge \varphi_{i+1}))$	✓	(f_1)	$\overline{\varphi}_i \vee \sigma_i \vee \delta_i$
		✓	(f_2)	$\overline{\varphi}_i \vee \sigma_i \vee \varphi_{i+1}$
	$\sigma_i \rightarrow \varphi_i$	✓	(g)	$\overline{\sigma}_i \vee \varphi_i$
	$\delta_i \wedge \varphi_{i+1} \rightarrow \varphi_i$	✓	(h)	$\overline{\delta}_i \vee \overline{\varphi}_{i+1} \vee \varphi_i$
	$\varphi_k \rightarrow \bigvee_{l \le i \le k} \sigma_i$		(i)	$\overline{\varphi}_k \vee \bigvee_{l \le i \le k} \sigma_i$

Table 1: Unfolding rationales and CNF clauses for LTL operators. A checkmark indicates that the clauses in the corresponding line must be instantiated over time ($0 \le i \le k$).

Using Theorem 1, we can determine for some unexpected counterexample or witness via $E_{WFM}(\varphi, \tau)$ or $E_{WFM}(\neg\varphi, \tau)$ the corresponding diagnoses in the specification's operators/subformulae [Pill and Quaritsch, 2013].

3 Exploiting a Specification's Parse Tree during Behavioral LTL Diagnosis

In this section we show how to exploit an LTL specification φ's actual parse tree in the search for diagnoses as of Theorem 1. Without loss of generality, we occasionally refer for our argumentation to the minimal conflicts that can characterize a diagnosis problem as of Reiter's diagnosis theory. The possible effects are discussed in the context of the well-known HS-DAG algorithm due to the easily graspable DAG that encodes its search. Our reasoning is however general enough, so that the underlying ideas transfer also to other diagnosis algorithms. The main observation our reasoning draws on is covered in Proposition 1.

Proposition 1. *If some subformula ψ from specification φ can resolve an issue (i.e. a minimal conflict), then so can all the superformulae of ψ.*

The correctness of this intuitive observation is easily shown considering our encoding for Theorem 1. When a subformula ψ is considered abnormal, the corresponding time-instantiated variables ψ_i are freed in that their values become undefined. Via the individual subformulae's encodings, the *chosen* values for ψ_i however still influence (depending on the operators) the evaluation of the variables of its superformulae (those formulae on the path from root vertex v_φ to v_ψ). Thus, if there is some satisfying assignment for ψ_i when considering ψ faulty, this assignment is still a satisfying one when considering a superformula δ to be at fault, and freeing the assignments of $\delta's$ subformulae (not signals!) that are irrelevant for the evaluation of φ anyway (due to $AB(\delta)$).

Our observation in Proposition 1 is also reflected in the minimal conflicts describing the diagnosis problem:

Proposition 2. *If a minimal conflict C_i contains some subformula ψ, then it contains also all its superformulae.*

The correctness of this proposition is easy to see. Superformula δ's not being in C would contradict Proposition 1 that δ can resolve/hit (at least) those minimal conflicts that ψ can resolve (including C_i). Note that while the proposition is obviously not true for a non-minimal C_i, even such a conflict might be pruned regarding subformulae where not all of its superformulae are in C_i (shedding some of the non-minimal/unnecessary components).

In the following we show how to use these propositions in order to derive new facts from already computed ones. That is, for instance, from some diagnosis Δ we can derive further diagnoses Δ'.

Lemma 1 (Infer-up)**.** *For a diagnosis $\Delta = \{\psi_1, \ldots, \psi_n\}$ and some $\psi_i \in \Delta$ with δ a superformula of ψ_i, the set $\Delta' = (\Delta \setminus \{\psi_j | \psi_j \in \Delta$ and δ is a superformula of $\psi_j\}) \cup \{\delta\}$ is a diagnosis as well.*

Proof. According to Proposition 1, a superformula δ of some ψ_i can resolve (hit) at least those conflicts that ψ_i can resolve. Thus replacing ψ_i with δ in some minimal diagnosis Δ constructs a set that can still resolve all conflicts. However, in order to derive a formal diagnosis (that per definition is subset-minimal), we have to remove all subformulae of δ from Δ, which, according to Proposition 1, is not a problem regarding resolved conflicts. □

This can help in the search space exploration, as approved hypotheses (diagnoses, or in the context of Reiter's algorithm consistent sets $h(n)$) might be converted easily into multiple ones. For the HS-DAG algorithm, for instance, we derive in Section 4 a corresponding strategy for expanding a node, labeling also a consistent node's siblings as consistent (without an explicit consistency check) if their edge labels refer to superformulae of the subformula at hand.

The following corollary describes the reasoning in the opposite direction; adding or replacing some ψ_i in Δ with one of its subformulae obviously cannot grow the set of C_is hit.

Corollary 1 (Infer-down)**.** *For some set $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that SD $\cup$ OBS $\cup \{\neg AB(c_i) | c_i \in$ COMP $\setminus \Delta\}$ is inconsistent, and a* subformula *δ of some $\psi_i \in \Delta$, the set $\Delta' = (\Delta \setminus \psi_i) \cup \{\delta\}$ is inconsistent as well.*

Considering this corollary, an HS-DAG strategy similar to the one above could be fathomed, inferring the inconsistency of a DAG node. For HS-DAG, the effects however would be hardly noticeable, due to the conflict cache that we consider standard in today's implementations. Retrieving a set not hit by $h(n) = \Delta'$ from the cache would always succeed, as an adequate one would have been registered previously for Δ (otherwise Proposition 2 would be violated). Thus we would not save an *expensive* theorem prover call.

The following variant of Corollary 1, where we do not replace ψ_i by subformula δ, but add δ to Δ, while valid for the same reasons, does allow us to prune the search space regarding subformulae.

Lemma 2 (Prune-down)**.** *For some set $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that SD $\cup$ OBS $\cup \{\neg AB(c_i) | c_i \in$ COMP $\setminus \Delta\}$ is inconsistent, and a* subformula *δ of some $\psi_i \in \Delta$, the set $\Delta' = \Delta \cup \delta$ is inconsistent. Furthermore Δ' and any of its supersets cannot be a diagnosis.*

Proof. Obviously, Δ' hits exactly those conflicts C_i also hit by Δ (according to Proposition 2), so that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta'\}$ cannot be consistent. Furthermore, by Proposition 1, ψ_i hits (at least) all the conflicts that δ hits, so that one could remove δ from Δ' and any of its supersets without affecting the set of conflict sets hit by them respectively. Thus neither Δ' nor any of its supersets can be a diagnosis that, per Def. 5, has to be subset-minimal. □

Regarding this lemma, we would like to note that HS-DAG perfectly implements this reasoning. That is, when retrieving a label for some non-leaf node n, it asks for some C_i not hit by $h(n)$. A corresponding C_i cannot contain a subformula of some $\psi_i \in h(n)$ due to Proposition 2.

Interestingly enough, the specific definition of a diagnosis allows us to consider some aspects of this reasoning also for superformulae, so that we can prune the search space also regarding superformulae.

Lemma 3 (Prune-up)**.** *For some $\Delta = \{\psi_1, \ldots, \psi_n\}$ such that $SD \cup OBS \cup \{\neg AB(c_i) | c_i \in COMP \setminus \Delta\}$ is inconsistent, adding some δ ($\Delta' = \Delta \cup \{\delta\}$) that is a superformula of some $\psi_i \in \Delta$ cannot yield a diagnosis.*

Proof. By Proposition 1 we know that δ hits (at least) all the conflicts that ψ_i hits, so that one could remove ψ_i from Δ' without affecting the set of hit conflict sets. As diagnoses have to be subset-minimal according to Definition 5, Δ' cannot be a diagnosis then. Obviously, only adding elements to Δ' cannot resolve the issue at hand, so that also no superset of Δ' can be a diagnosis. □

The effect of this lemma is that when some part of a solution is established (e.g. during a structured conflict-driven search with HS-DAG, or for a partial assignment when computing diagnoses directly with a SAT-solver), we can rule out all superformulae of any ψ_i already considered up to the point where we remove ψ_i again (e.g. during some backtracking step in the SAT-solver).

Summarizing, our lemmas allow us to dynamically adapt and focus the search for diagnoses with easily derived positive or negative data. In the next section we discuss the adoption of our reasoning in the context of the well-known HS-DAG algorithm.

4 HS-DAG

For the HS-DAG algorithm we implemented the reasoning from Lemmas 1 and 3 in the procedures INFERUP and PRUNEUP covered by Algorithms 2 and 1, respectively. Please note that HS-DAG covers the reasoning from Lemma 2 by construction, as mentioned before.

Prior to the expansion process of an (inconsistent) HS-DAG node, we call the procedure PRUNEUP. It discards from n's intended label the superformulae of all $\psi_i \in h(n)$. As conflicts may comprise multiple (overlapping) "chains" to the parse tree's root, we mark those superformulae in $\mathcal{T}$ which have been examined already.

The label $\ell(n)$ of a node n is either a conflict, "✓" (consistent), "×" (closed) or "(yet) undefined". We assume now that HS-DAG expands an inconsistent node by iterating over its label/conflict C, considering first those subformulae farthest from v_φ in the parse tree $\mathcal{T}$.

Whenever a node n is found to be consistent (i.e. $h(n)$ is "consistent"), under certain conditions INFERUP labels siblings whose incoming edges are labeled with superformulae

Algorithm 1 Pruning conflict sets in HS-DAG.

Requires: n — HS-DAG node being expanded
1: **procedure** PRUNEUP(n, $\mathcal{T}$):
2: CLEARMARKS($\mathcal{T}$)
3: $C \leftarrow \ell(n)$
4: **for all** $\psi \in h(n)$ **do**
5: **while** $\psi \neq$ NULL $\wedge$ ψ not marked **do**
6: $\psi \leftarrow$ PARENT($\psi, \mathcal{T}$)
7: $C \leftarrow C \setminus \{\psi\}$
8: mark ψ
return C

Algorithm 2 Inferring new diagnoses in HS-DAG.

Requires: n — consistent HS-DAG node
Requires: ψ — subformula that led to n
1: **procedure** INFERUP(n, $\mathcal{T}$, ψ):
2: $C \leftarrow \ell$(PARENT(n)) ▷ parent node's conflict
3: $\delta \leftarrow$ PARENT($\psi, \mathcal{T}$) ▷ parent in the parse tree
4: **while** $\delta \neq$ NULL **do**
5: **if** δ is not marked $\wedge$ $\delta \in C$ **then**
6: $n' \leftarrow$ GETSIBLING($(h(n) \setminus \psi) \cup \{\delta\}$)
7: **if** $\exists m$ **s.t.** $h(m) \subseteq h(n') \wedge \ell(m) =$ ✓ **then**
8: $\ell(n') \leftarrow \times$
9: **else**
10: $\ell(n') \leftarrow$ ✓
11: mark δ
12: **else**
13: **break**
14: $\delta \leftarrow$ PARENT($\delta, \mathcal{T}$)

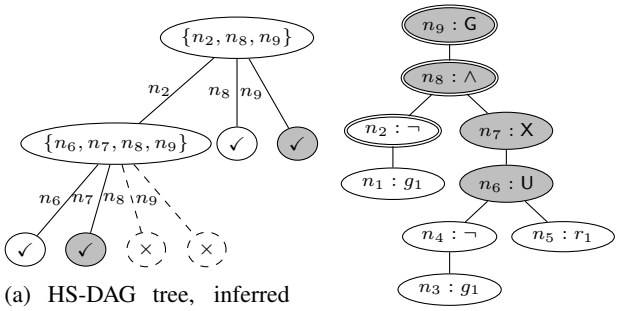
with ✓ too. In lines 7 to 8 we make the corresponding HS-DAG subset check whether there is a subset in $h(n')$ that is a diagnosis, checking whether n' should be closed. Again, we stop following a path to the parse tree's root whenever we encounter a superformula already considered.

The effects of our reasoning become evident in the following two examples. Our first example is adopted from [Pill *et al.*, 2006] and was also diagnosed in [Pill and Quaritsch, 2013]. It features a two line arbiter with request lines r_1 and r_2 and the corresponding grant lines g_1 and g_2. Its specification consists of the following four requirements: R_1 demanding that requests on both lines must be granted eventually, R_2 ensuring that no simultaneous grants are given, R_3 ruling out any initial grant before a request, and finally the faulty $R_4 : \forall i \in \{1, 2\} : \mathsf{G}\,(g_i \rightarrow \mathsf{X}\,(\neg g_i \,\mathsf{U}\, r_i))$ preventing additional grants until new incoming requests. Testing her specification, a designer defines an unexpectedly failing witness (i.e. a trace that should satisfy the specification but violates it) $\tau = \tau_0\tau_1(\bot)^\omega$ featuring consecutive (and instantly granted) single requests for both lines:

$\tau_0 = \quad r_1 \wedge \quad g_1 \wedge \neg r_2 \wedge \neg g_2$
$\tau_1 = \neg r_1 \wedge \neg g_1 \wedge \quad r_2 \wedge \quad g_2$

As already pointed out in [Pill *et al.*, 2006], the problem in this specification is the until operator $\neg g_i \,\mathsf{U}\, r_i$ in R_4 that should be replaced by its *weak* version $\neg g_i \,\mathsf{W}\, r_i$: While the idea of both operators is that $\neg g_i$ should hold until r_i holds, the *weak* version does not require r_i to hold eventually, while the "strong" one does. Thus, R_4 in its current form repeatedly requires requests that are not provided by τ, and which is presumably not in the designer's intent.

Our standard HS-DAG implementation, as used also in [Pill and Quaritsch, 2013], obtained for this scenario 31



(a) HS-DAG tree, inferred nodes shaded, pruned "edges" dashed.

(b) Parse tree, the two conflicts are shaded/double lined.

Figure 1: HS-DAG run for the arbiter formula.

diagnoses, including the one pinpointing to wrong *until* operators in both instances of R_4. It issued 34 theorem prover (SAT solver) calls in total, when building its DAG comprising 44 nodes (cf. Table 2).

The effects of our optimizations can be seen in Table 2. While the number of nodes constructed by HS-DAG could be reduced from 44 to 38 (some minimum number of nodes is needed to represent the 31 diagnoses) using PRUNEUP, the number of consistency checks that require a theorem prover call could be cut down from 31 to 13 (-58%) via 18 diagnoses inferred using INFERUP. This resulted also in a run-time reduction (over 100 runs) of more than 46% even for this simple example. Using the PRUNEUP node-reduction alone resulted in a slight but negligible (<1%) run-time penalty. Using INFERUP and PRUNEUP aggregates the advantages, offering fewest nodes as well as an attractive run-time.

For visualizing the effects of our INFERUP and PRUNEUP optimizations, we extract an even smaller example from the arbiter requirement R_4. As PRUNEUP only affects DAG levels with $|h(n)| > 1$, we purposefully inject a second fault in R_4 by replacing the logic OR (in the rewritten implication) with a logic AND: $\varphi = \mathsf{G}\,(\neg g_1 \wedge \mathsf{X}\,(\neg g_1\,\mathsf{U}\,r_1))$ and consider a single line. The trace consists of a single request and grant: $\tau = \tau_0(\bot)^\omega$ with $\tau_0 = r_1 \wedge g_1$. Figure 1 depicts the DAG and parse tree for this example.

We start expanding the root node using n_2 (inconsistent) and n_8. As $\{n_8\}$ is consistent, we can infer $\{n_9\}$ to be consistent as well, as n_9 is a superformula of n_8 (see Figure 1b). For the node labeled $\{n_6, n_7, n_8, n_9\}$, we can skip n_8 and n_9 in the expansion as their subtrees generate supersets of $\{n_8\}$ and $\{n_9\}$ only (dashed edges/nodes). Similar to the level above, we can infer $\{n_2, n_7\}$ to be a diagnosis due to $\{n_2, n_6\}$ being a diagnosis and n_7 a superformula of n_6.

	–	P↑	I↑	P↑ + I↑
# HS-DAG nodes	44	38	44	38
# TP consistency checks	31	31	13	13
# TP conflict computations	3	3	3	3
# pruned "edges"	–	6	–	6
# inferred diagnoses	–	–	18	18
# diagnoses	31	31	31	31
run-time (sec.)	0.3801	0.3821	0.2029	0.2033

Table 2: HS-DAG statistics for the arbiter example using no/PRUNEUP/INFERUP/PRUNEUP+INFERUP optimization

4.1 Experimental Results

We applied our optimizations to the Python (CPython 2.7.1) implementation used in [Pill and Quaritsch, 2013]. We ran our tests on an early 2011-generation MacBook Pro (Intel Core i5 2.3GHz, 4GiB RAM, SSD) with an up-to-date version of Mac OS X 10.6, the GUI and swapping disabled, and using a RAM-drive for the file system.

As test samples, we generated random LTL formulae as suggested in [Daniele *et al.*, 1999] with $N = \lfloor |\varphi|/3 \rfloor$ variables and a uniform distribution of LTL operators. We injected *triple* faults in order to derive φ_m from φ, and using our LTL encoding, we derived an assignment for $\tau \wedge \varphi \wedge \neg\varphi_m$ that defines τ for $k = 100$ and $l = 50$. We verified that φ_m is a valid diagnosis considering φ and τ.

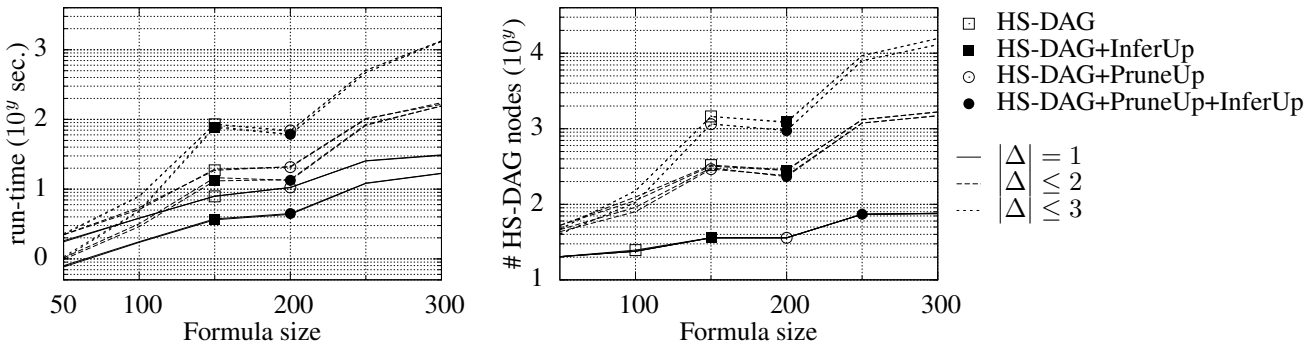
To obtain the results in Figure 2, we generated 10 random diagnosis problems as outlined above for any $|\varphi|$ in $\{50, 100, \ldots, 300\}$, ran HS-DAG ten times with a diagnosis cardinality limit of 1, 2 and 3 (single, double and triple faults) with our various optimizations applied, and plotted average values. For the single fault diagnosis runs (solid lines), we observe a run-time reduction of up to approx. 60% due to INFERUP. The run-time benefit diminishes with rising maximum diagnosis cardinality, when, intuitively, the number of diagnoses (and thus inferable nodes) grows slower than the total number of DAG nodes. While PRUNEUP shows virtually no influence on the run-time, Figure 2b depicts its impact on the number of DAG nodes constructed for a specific problem. Growing with rising maximum diagnosis cardinality, a reduction of up to 23% was possible for $|\varphi| = 200$ and $|\Delta| \leq 3$. We thus argue that PRUNEUP eliminates HS-DAG nodes that would have been closed otherwise by subset-checks later-on.

Summarizing, while PRUNEUP could achieve a significant DAG node reduction for large diagnosis cardinalities only (i.e., in unbounded runs), INFERUP could substantially reduce run-times for the more practical, low-bound case.

5 Discussion and Conclusions

With us focusing on LTL specification diagnosis, similar ideas have been exploited previously for other domains. For example, in the context of circuit diagnosis, the concepts of dominators and cones are exploited for circuit abstraction and a diagnosis speed up. Originating in the field of program analysis using control flow graphs [Prosser, 1959; Lengauer and Tarjan, 1979] and later adopted for the analysis of digital circuits [Kirkland and Mercer, 1987], a dominating component can "overrule" the dominated ones (referred to as its corresponding *cone*) because, e.g., it is "closer to the output". As dominators for an arbitrary graph structure can be computed in linear time [Buchsbaum *et al.*, 2008], approaches such as [Siddiqi and Huang, 2007; Metodi *et al.*, 2012] focus their diagnostic search on those gates first. The resulting *top level diagnoses* are then refined by creating further potential diagnoses with dominators replaced by gates from their cone.

While cones are not directly exploitable for specification diagnosis (we would get a single (maximal) cone if applied to a static LTL parse tree or raise complexity unnecessarily when temporally unfolding it) [Mangassarian *et al.*, 2011; Le *et al.*, 2012] peruse the notion of (reverse) dominance for their SAT-based RTL debugging, resulting in implied (non-)solutions. We showed that for consistency-based diagnosis using HS-DAG, we can achieve a speed-up of about



(a) Run-time

(b) Number of HS-DAG nodes

Figure 2: Diagnosis performance for random samples.

factor two also for our problem domain, using implied (inferred) solutions. On the other hand, we showed that the implication of non-solutions does not speed up HS-DAG's diagnosis process due to the usage of a conflict set cache. Instead, we could optimize HS-DAG's search strategy in the context of a domination relation by pruning the conflicts depending on the current tree context. The latter resulted in up to 23% fewer HS-DAG nodes for our tests.

We expect our reasoning to be attractive also for similar, (temporal) formula-based descriptions. Future work will include the transfer of our search strategy optimizations to the direct diagnosis computation with SAT/constraint solvers.

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): P22959-N23 ("MoDiaForTeD"). The authors would like to thank Franz Wotawa for fruitful discussions and the anonymous reviewers for their valuable comments.

References

[Buchsbaum *et al.*, 2008] A. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. Tarjan, and J. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.

[Daniele *et al.*, 1999] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for Linear Temporal Logic. In *Computer Aided Verification*, pages 249–260, 1999.

[Fisman *et al.*, 2009] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M.Y. Vardi. A framework for inherent vacuity. In *International Haifa Verification Conference on Hardware and Software: Verification and Testing*, pages 7–22, 2009.

[Greiner *et al.*, 1989] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artif. Intelligence*, 41(1):79–88, 1989.

[Kirkland and Mercer, 1987] T. Kirkland and M. R. Mercer. A topological search algorithm for ATPG. In *ACM/IEEE Design Automation Conference*, pages 502–508, 1987.

[Kupferman, 2006] O. Kupferman. Sanity checks in formal verification. In *International Conference on Concurrency Theory*, pages 37–51, 2006.

[Le *et al.*, 2012] B. Le, H. Mangassarian, B. Keng, and A. Veneris. Non-solution implications using reverse domination in a modern SAT-based debugging environment. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 629–634, 2012.

[Lengauer and Tarjan, 1979] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–141, 1979.

[Mangassarian *et al.*, 2011] H. Mangassarian, A. Veneris, D. E. Smith, and S. Safarpour. Debugging with dominance: On-the-fly RTL debug solution implications. In *International Conference on Computer-Aided Design*, pages 587–594, 2011.

[Metodi *et al.*, 2012] A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling model-based diagnosis to Boolean satisfaction. In *AAAI Conference on Artificial Intelligence*, pages 793–799, 2012.

[Pill and Quaritsch, 2013] I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *International Joint Conference on Artificial Intelligence*, pages 1053–1059, 2013.

[Pill *et al.*, 2006] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In *Conference on Design Automation*, pages 821–826, 2006.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Prosser, 1959] R. T. Prosser. Applications of Boolean matrices to the analysis of flow diagrams. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 133–138. ACM, 1959.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Siddiqi and Huang, 2007] S. Siddiqi and J. Huang. Hierarchical diagnosis of multiple faults. In *International Joint Conference on Artificial Intelligence*, pages 581–586, 2007.

[Wiegers, 2001] K. E. Wiegers. Inspecting requirements. *StickyMinds Weekly Column*, July 2001. http://www.stickyminds.com.

On the Role of Model-based Diagnosis in Functional Safety

Michael Hofbaur[1] and **Martin Sachenbacher**[2]
[1]UMIT, Institute of Automation and Control Engineering
Hall i.Tyrol, Austria
e-mail: michael.hofbaur@umit.at
[2]TU München, Department of Informatics
Garching, Germany
e-mail: sachenba@in.tum.de

Abstract

Functional safety is an integral part of many modern technical systems. It comprises design processes and operational measures for safety-critical systems, such that the risk of physical injury or damage to the health of people or the environment is reduced to an acceptable level. The importance of assuring and certifying a particular safety integrity level is reflected by recent international standards for functional safety, such as IEC 61508 and ISO 26262. However, it turns out that safety mechanisms based on on-line monitoring play only a minor role in these contemporary standards, which may be somewhat surprising or disappointing from the perspective of model-based diagnosis research, since it claims to offer generic and powerful tools to detect and mitigate faults in technical systems. In this paper, we give an overview of functional safety concepts and its related industry standards. From this, we can infer requirements that model-based methods must meet in order to be applicable in the context of functional safety. We also highlight possible solution approaches from model-based diagnosis research that are able to satisfy the strong constraints imposed by functional safety standards.

1 Introduction

Many modern technical systems pose a risk to harm people or the environment. As a consequence, engineers are obliged to put much effort into designing these systems such that the nominal operation is safe and possible malfunctions will not endanger the environment beyond a level that is acceptable within our society. Ongoing health monitoring, fault detection and reconfigurable/robust control is thus standard in many safety-critical applications. The area of model-based diagnosis (MBD) claims to provide a comprehensive toolkit of modeling formalisms and generic solvers for system monitoring and diagnosis. Nevertheless, most real-world technical systems instead employ special-purpose – and to our research-community's understanding, often less powerful – monitoring and diagnosis mechanisms to implement this critical functionality.

Contemporary industry standards for functional safety, such as IEC 61508 [iec, 2010] and its 2011 adaptation ISO 26262 [iso, 2011] for automotive systems, provide the essential safety framework for addressing possible hazards of electronic and electrical systems. They define requirements for the entire life cycle of an artifact, including design, manufacturing, operation, service, and decommission. The aim of these standards is to properly identify all critical operational and fault situations for a specific sub-system or the system itself, and to provide means for avoiding systematic faults within the design, production and operation as well as safely coping with random hardware faults during operation.

Table D.2 — Systems

Safety mechanism/measure	See overview of techniques	Typical diagnostic coverage considered achievable	Notes
Failure detection by on-line monitoring	D.2.1.1	Low	Depends on diagnostic coverage of failure detection
Comparator	D.2.1.2	High	Depends on the quality of the comparison
Majority voter	D.2.1.3	High	Depends on the quality of the voting
Dynamic principles	D.2.2.1	Medium	Depends on diagnostic coverage of failure detection
Analogue signal monitoring in preference to digital on/off states	D.2.2.2	Low	—
Self-test by software cross exchange between two independent units)	D.2.3.3	Medium	Depends on the quality of the self test

Figure 1: Table D.2 from Part 5 of the ISO 26262(2011) international standard, showing safety mechanisms/measures and their typical diagnostic coverage considered achievable

Model-based diagnosis that focuses on on-line systems monitoring during operation is thus able to cover only a fraction of the overall functional safety framework. We believe it is worthwhile to understand the entire standards and their implications, in order to obtain a solid understanding of the industry requirements and their implications for current research directions in model-based diagnosis. For example, the ISO 26262 standard lists several state-of-the-art mechanisms for ensuring system integrity. While MBD claims to provide very powerful mechanisms to detect and localize even subtle faults from their complex system-wide implication, the ISO standard classifies measures of the kind *failure detection by on-line monitoring* to provide a *typical diagnostic coverage considered achievable* of level *low* only (first line of table in Fig. 1)! As a consequence, unless we are able to prove that model-based diagnosis provides the required level of diagnostic coverage and satisfies the strong requirements imposed in the design stage of a system to ensure functional safety, we will not be able or even be allowed to apply MBD methods in industrial safety-critical systems.

The main goal of this paper is therefore to draw the attention of academic research onto this highly relevant topic, and to provide the foundation for a possible bridge between

the two partly conflicting worlds. The following Section 2 will thus provide a comprehensive overview on Functional Safety and its associated industry standards. We particularly highlight aspects of functional safety that touch the MBD domain and thus impose constraints on its applicability. Given this overview, Section 3 provides a short analysis of the implications for MBD, both for modeling frameworks and algorithmic approaches. Section 4 will then focus on specific diagnosis schemes and identify some possible solution approaches that utilize results from MBD research, whilst at the same time are likely to satisfy the strong constraints imposed by the functional safety standards.

2 Functional Safety

In many modern systems, a mechatronic approach to systems design is now almost standard. This means one obtains a system's behavior through advanced control and automation, which orchestrates the individual system components according to the high demands on performance, robustness, and efficiency. This coordinated interplay of the individual system components, where most interactions are encoded in control/software, is thus very vulnerable to faults and failures. Whereas hardware components fail in rather predictable ways, software crashes in a very unpredictable manner. Furthermore, the system's complexity with its many modes of operation can lead to undesired behavior due to overseen/unplanned interactions within the system components. These malfunctions can lead to unsafe operational situations for humans and the environment.

The goal of Functional Safety is to reduce the risk of physical injury or damage to people or the environment down to an acceptable level. As a consequence, it is essential to build a system in such a way that malfunctions with dangerous consequences are avoided as good as possible. This includes several aspects. Firstly, one has to consider both random (hardware) faults and systematic (design) faults, and thus adapt the design and production process accordingly. Secondly, one has to consider all operational situations within the life cycle of a system. A car, for example, can impose dangerous situations during operation, but also during service or decommission.

International Functional Safety standards thus put a large emphasis on systems design, but include also requirements for the other stages in the life cycle of an engineering artifact. Such standards are typically not mandatory, which means there exists no direct legal requirement to comply with a certain safety standard. However, the fulfillment of the respective standard, certified by an independent agency, is often a strong argument in the case of liability claims. In addition, it frequently occurs that OEMs demand compliance with the standard from their part suppliers. A general Functional Safety standard is IEC 61508 [iec, 2010], which is applied to the development of electrical, electronic and programmable electronic safety-critical systems and forms the basis for many domain specific standards. Specifically, ISO 26262 [iso, 2011] is the adaptation of IEC 61508 to the automotive domain. These comprehensive standards also define their specific nomenclature. The following subsections will therefore provide a compact summary, focusing in particular on ISO 26262. This contemporary standard consists of nine normative parts (Part 1–9) and a guideline.

2.1 Hazard Analysis, Risk Assessment, and Functional Safety Concept

As a first and major step in the Functional Safety-enriched design process, one performs a detailed hazard analysis and risk assessment of a specific device or sub-component under development. The standard uses the term *item* for this functional entity of interest. The hazard and risk analysis considers a matrix of all possible operational conditions and all possible malfunctions, and determines the safety consequences in an exhaustive way. The result is a classification of all possible hazardous events in terms of the required safety integrity levels[1] (ASIL A - ASIL D) as well as a specification of associated safety goals in terms of functional objectives including the required reaction times to safety critical malfunctions. A safety integrity level is a targeted level of risk reduction, with ASIL A being the lowest and ASIL D the highest risk class. A consecutive concept development phase derives functional safety requirements from the safety goals and defines preliminary architectural measures w.r.t. fault detection and failure mitigation, transitioning to a safe state, fault-tolerance mechanisms and functional redundancies, as well as emergency operations (ISO 26262, part 3).

In a sense this process can be seen as a bottom-up approach where one identifies hazardous operational conditions, infers their associated root source, and derives objectives for their detection and mitigation (safety goals). It is the purpose of a following systems development process to implement these objectives through appropriate means along with the original functionality of the system.

2.2 Design, Implementation, and Validation

The parts 4 – 6 of the ISO-26262 standard define means for the system's design, hard- and software development, system's implementation and the verification and validation process. They define an integrated design process that considers both, (i) the requirement specifications for the system's functionality and (ii) the functional safety objectives/requirements. Functional safety is thus a fundamental part of the system's design process that is integrated within the architectural considerations and control functionality from the beginning. The mandatory design process implements a sophisticated engineering approach for system's development that is based on the V-model reference process from software engineering and the model-based design paradigm. The motivation for such a formalized approach is to adequately cope with complexity of the system and the development process, to explore the mechatronic design space as good as possible and to actively avoid systematic faults.

Concept Phase (ISO 26262 Part 4)

The concept phase establishes the general system architecture for the item under development. The system's complexity is dealt with by hierarchical design, modularization, utilization of well-trusted design patterns/principles and a comprehensive hardware/software interface specification. These design principles are suggested in order to reduce the risk of system malfunctions due to systematic faults or due to flaws within the design process. A decision not to re-use well-trusted design principles and to de-

[1] Automotive Safety Integrity Level (ASIL), as ISO 26262 defines application-specific means for functional safety for road vehicles.

sign new and novel elements for an item should be justified. Thus it is compulsory to prove superiority over a standard, well-trusted approach.

Detailing the system's requirements in terms of hard- and software functionality, one can allocate the safety objectives to individual hardware and software components and refine them into technical safety requirements for the consecutive hardware/software design. As a consequence, one obtains a safety-enriched architecture where specific means for malfunction detection are integrated within the overall functionality of the system to provide safety as an active part of system's control, not just as a "prevent failures" task. This seamless integration provides a detailed specification of dedicated hard- and software functionality and their safety-related interplay that is detailed within the comprehensive hardware-software specification.

The system design is supported through model-based techniques that allow preliminary simulation studies and an exhaustive deductive and inductive analysis, e.g. fault-tree analysis (FTA) and failure mode and effects analysis (FMEA), respectively. Thus, as the architecture becomes more and more explicit, one can translate possible malfunctions into associated random hardware faults within individual architectural components and specify hardware diagnostic features and their utilization by the control software for appropriate fault handling.

Fault classes and terminology in ISO 26262

Prior to detailing the individual aspects of hardware development, it is useful to introduce the different fault classes for hardware elements and their characterization as used within the context of the ISO standard (Figure B.1 and B.2 in ISO 26262 Part 5). The standard uses the established notion to denote the (i) abnormal condition that can cause an element or item to fail as *fault*, (ii) the discrepancy between the computed/observed/measured value or condition and the true/specified/correct value/condition as the *error*, and (iii) the finally possible termination of the ability of an element to perform a function as required as the *failure*. Faults, however, are detailed as follows:

- *Safe Fault:* (i) Failure mode of a non safety-related hardware element or (ii) a fault in a safety-related hardware element that does not cause a violation of a safety goal by itself or in combination with another independent failure and does not require a dedicated safety mechanism. An associated system's analysis assigns the probability λ_S.
- *Single-Point Fault:* Failure mode with the potential to violate a safety goal which is neither detected nor covered through a dedicated safety mechanism. (λ_{SF})
- *Residual Fault:* Failure mode with the potential to violate a safety goal despite the fact that it is detected through a dedicated safety mechanism. (λ_{RF})
- *Detected Multiple-Point Fault:* Detected and appropriately controlled safety-relevant multiple failure mode. (λ_{MPFD})
- *Perceived Multiple-Point Fault:* Perceived (through the driver) and appropriately controllable safety-relevant multiple failure mode. (λ_{MPFP})
- *Latent Multiple-Point Fault:* Undetected safety-relevant multiple-point failure mode that results in a violation of a safety goal. (λ_{MPFS})

Metric	ASIL B	ASIL C	ASIL D
single-point fault	$\geq 90\%$	$\geq 97\%$	$\geq 99\%$
latent-fault	$\geq 60\%$	$\geq 80\%$	$\geq 90\%$

Table 1: Required levels for fault metrics for ASIL classes (Tables 4 and 5 of ISO 26262 Part 5)

	ASIL B	ASIL C	ASIL D
random failure target value	$< 10^7\ h^{-1}$	$< 10^7\ h^{-1}$	$< 10^8\ h^{-1}$

Table 2: Probability levels for safety-goal violation due to hardware failures (Table 6 of ISO 26262 Part 5)

The probability values are specified for all safety-relevant hardware elements of an item individually. The standard then provides a method to deduce two robustness metrics: the *single point fault metric* defines the robustness of an item w.r.t. single-point and residual faults, whereas the *latent fault metric* defines the robustness of an item w.r.t. multi-point faults, that is latent faults in particular (ISO 26262 Part 5, Annex C). Depending on the ASIL level for an item, it is then required the reach the specific levels for these metrics shown in Table 1.

The exhaustive hazard analysis and risk assessment that provide the safety-goals for the system and the derived technical safety requirements also have to specify important timing information for the associated safety mechanisms, in particular:

- *Fault Tolerant Time Interval (t_{FTI}):* Time span between the fault incident and the earliest resulting hazardous event/failure.
- *Fault Reaction Time (t_{RT}):* Time span for executing a safety-mechanism that includes the detection of a fault and the transition to a safe state.
- *Diagnostic Test Interval:* Following the definitions above, it is necessary to schedule a safety mechanism at least with sampling rate $T_D = 1/(t_{FTI} - t_{RT})$.

Given that one controls a dynamic system and the system takes (in a particular fault situation) t_{t2ss} seconds to transition to a safe state, one needs to devise a diagnostic procedure that is capable to detect and identify the fault within $t_{FTI} - t_{t2ss}$ seconds.

Hardware Development (ISO 26262 Part 5)

The main focus in hardware development is to build a reliable electrical and electronic subsystem for an item. To achieve the required high levels of system integrity (Table 2), one has to utilize well-trusted architectural schemes, such as redundancy, and the system's ability to monitor its health state and to mitigate faults within its individual components as good as possible. Again, a well-structured development process aims for minimizing systematic failures and utilizes an exhaustive reliability analysis to verify demanding requirements for the hardware's reliability to minimize the impact of faults in individual elements. Diagnosis and health monitoring is thus an integrated part of hardware development and the resulting system includes additional elements to support the system's ability for self-monitoring and diagnosis. Whereas random faults in individual components are a major focus of analysis, design and verification, it is also important to consider non-functional sources

for failures, such as vibration, temperature, EMI, cross-talk, etc. within the development process.

Safety analysis for the detailed hardware design includes the evaluation of the *diagnostic coverage (DC)*, i.e. the proportion of faults that a dedicated system's design and an additional safety mechanism can appropriately deal with. The ISO 26262 standard provides in Part 5 Annex D an informative listing of various mechanisms and techniques (e.g. the initially cited table D.2 of Fig. 1) and provides typical diagnostic coverage levels for them. The *low*, *medium* and *high* diagnostic coverage levels correspond to 60 %, 90 % and 99 %, respectively. Annex D is classified as *informative* and not *normative*. As a consequence, one can claim a higher diagnostic coverage for a particular implementation of a mechanism based on a detailed analysis.

We illustrate the principles given above with a well-trusted hardware design used for automotive electronic power control [Pfeufer *et al.*, 1999], shown in Fig. 2.

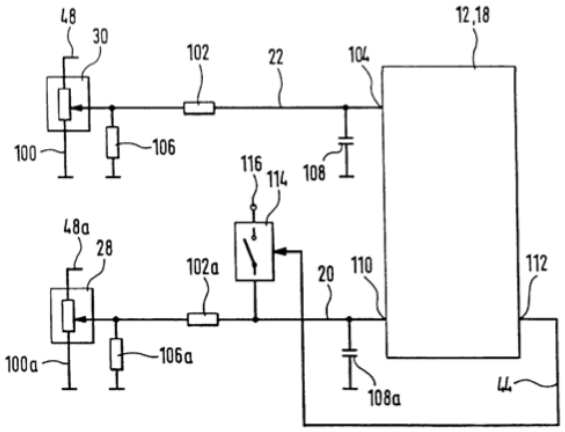


Figure 2: Circuit schematic of a fault-tolerant electronic gas pedal from US Patent 5.995.885

The item/system e-gas uses a fault tolerant gas pedal that contains two complementary sensors for the gas pedal position. It realizes several architectural features for fault detection and mitigation:

- *Sensor Valid Range (26262-5 Table D.11 / DC: Low)*: Both sensors (elements 30 and 28 in Fig. 2) do not utilize the full range for measuring the gas pedal position. In idle position, sensor 1 provides $v_{min,1} > 0$ and sensor 2 provides the complementary value of $v_{max,2} < V_{sup}$, where V_{sup} denotes the supply voltage of sensor 1 and 2 (elements 48, 48a). At full-gas position one obtains analogously: $v_{max,1} < V_{sup}$ and $v_{min,2} > 0$, respectively. The valid ranges are typically different for both sensors as well, e.g. $v_{max,1} > v_{max,2}$ and $v_{min,1} > v_{min,2}$. Shorts to zero or V_{sup} would provide out-of-range readings that can be easily detected.
- *Sensor Correlation (26262-5 Table D.11 / DC: High)*: The redundant sensor reading, where voltage at sensor 1, v_{s1}, increases with increasing gas pedal position, whereas v_{s2} decreases in the opposite sense provides complementary values that can easily be checked for consistency during on-line operation. This principle allows one to identify in-range faults such as drifts, offset and stuck-at faults.
- *Dynamic Principles (26262-5 Table D.2 / DC: Medium)*: The electronic switch (114) allows the MCU (12.18) to periodically connect sensor line 2 (20) to a fixed out of range voltage (116), e.g. V_{sup}. Measuring line 2 one has to observe this controller-induced dynamic change. This operation allows one to check the A/D converter channel used in the MCU for sensor 2. Refined instances of this principle use (MCU-controlled) dynamically changing supply voltages for the sensors (e.g. a superposition of V_{sup} with a specific voltage pattern), and check the sensor readings for this signature.
- *Schematic Features*: The pull-down resistors (106, 106a) ensure out-of-range sensor readings whenever a sensor becomes disconnected. Resistors (102, 102a) and the capacitors (108, 108a) provide a low-pass filtering of the sensor signal to suppress high-frequency noise (EMI) on the sensor cable.

Several other features and methods are usually applied in addition to this simplified hardware architecture. We refer the interested reader to [Pfeufer *et al.*, 1999] or [Schäuffele and Zurawka, 2013]. The example is typical for safety-instrumented systems developed along the ISO norm, in the sense that specific hardware measures for diagnosability are built into its design, and diagnosis is performed actively rather than just passively interpreting the available sensor readings.

Software Development (ISO 26262 Part 6)

The standard also regulates the development of software components in order to ensure the safety goals. This includes requirements both for the overall software architectural design and the design and implementation of individual software units. A key concern is the traceability of requirements and design decisions during the process, and the possibility to perform testing and verification at different stages of the software development process; the higher the ASIL, the more rigorous these analysis methods have to be.

During the *initiation of software development* phase, an overall plan of the software development process must be prepared, in particular including all tools and languages that will be used throughout software development. For any tools used, the standard demands their assessment in the form of a *tool confidence level (TCD)* metric, which is composed of a tool impact value ("the possibility that a malfunction of a particular software tool can introduce or fail to detect errors in a safety-related item or element being developed") and a tool error detection value ("the confidence in measures that prevent the software tool from malfunctioning and producing corresponding erroneous output, or in measures that detect that the software tool has malfunctioned and has produced corresponding erroneous output"). The standard also demands proficiency of the personnel involved in the tool-chain.

In the *safety requirements specification* phase, all software-based functions must be identified which could potentially impact safety goals by either directly producing an unsafe state, or by failing to correctly handle a hardware or software fault. The requirements for each software component also include timing requirements and the interface between the component and other system components. A high-level design for each software component is then developed in the *architectural design* phase, and the confor-

mance of the architectural design to the safety requirements must be verified.

In the subsequent *software unit design and implementation* phase, each subsystem is designed and implemented. One particular requirement of the standard is that it demands a low code complexity, and only a rather limited number of language constructs is allowed in the final code, as shown for instance in Fig. 3).

Table 8 — Design principles for software unit design and implementation

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a,b]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[a,b]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[a,b,c]	++	++	++	++
1j	No recursions	+	+	++	++

[a] Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.
[b] Methods 1g and 1i are not applicable in assembler programming.
[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

Figure 3: Table 8 from Part 6 of the ISO 26262(2011) international standard, showing required principles for software unit design and implementation

Once the code is implemented, *unit testing* and *integration testing* phases must be passed. The goal of unit testing is to individually test the correctness of each low-level software module, whereas the purpose of integration testing is to validate software systems comprised of several modules. Since safety functions are treated as an integral part of the system, this task also applies to any code that is meant to detect and isolate faults in the system. The standard in particular recommends function coverage (every function must be called at least once) and call coverage (every call must be issued at least once) for ASIL A and B, and highly recommends it for ASIL C and D. Although the standard includes an explicit clause for legacy parts that have already been used in the field, for software components this typically does not apply, because even extensive operation does not necessarily guarantee the requested formal code coverage. The last phase is the *safety requirements verification*, which must be performed in order to ensure that the software works correctly in its target environment. ISO 26262 recommends semi-formal methods such as fault simulation runs for all levels, and formal model-based verification methods for ASIL C and higher.

System Integration, Verification & Validation (ISO 26262 Part 4)

System's design and (domain-specific) development is done with testability in mind. This includes tests for health monitoring and diagnosis during operation, but also for verification and validation purposes during the system integration phase of the development process. Given the system's specification, and in particular, the specification of the safety goals, it is mandatory to plan the verification process during design and perform it accordingly during the system integration phase. As the system evolves in terms of maturity of its individual elements/parts, one applies model-based techniques (software in the loop, hardware in the loop) to verify the design goals and in particular also the safety objectives and the obtained diagnostic coverage of the safety mechanisms. A consistent and sufficiently detailed/accurate model that reflects the system as it evolves from concept to realization is thus crucial for avoiding systematic faults in the design process. Thus the standard requires exhaustive back-to-back tests that compare the responses of system and/or its sub-systems with the model predictions for an exhaustive set of stimuli to verify the model or models that are used throughout the development process.

3 Implications for Model-based Diagnosis

Model-based diagnosis is a research direction that aims at providing theoretical results in the form of modeling formalisms and algorithmic frameworks, as well as practical tools for detecting, isolating, identifying, and possibly mitigating faults in a system. In this section, we review the role that MBD can possibly play and gaps that need to be bridged in the context of Functional Safety and its contemporary standards, where the primary goal is to reduce risks of a safety-critical system to acceptable levels.

Implications on Modeling Formalisms and Processes

The model-based approach to system's engineering and the requirement to verify and validate the safety mechanisms ensures models at adequate level of detail to serve as the basis for MBD methods. A model is still just an abstraction of the real world, and a model-based safety mechanism thus always poses the risk of uncovered operational and fault situations. The MBD consistency-based approach to health monitoring and diagnosis, however, provides an interesting concept for functional safety – the *unknown mode*. I.e. the diagnoser can identify situations where even a high fidelity model does not explain the behavior seen anymore and thus could trigger a dedicated safety mechanism / procedure.

High confidence on dynamic methods suggest to (i) use hybrid health monitoring mechanisms and possibly (ii) active diagnosis methods to track a system along a dynamic path (within its operational range) and thus utilize dynamic principles as major concept for on-line monitoring, fault detection and diagnosis.

However, building high-fidelity hybrid models for a mechatronic system is non-trivial and standard design models (e.g. for control design) might not be sufficiently detailed for MBD diagnosis methods. In order to ease the application of MBD methods we have to provide tools for engineers so they can easily enrich their control-based models for diagnosis, and apply methods from the MBD toolkit with high confidence more straightforwardly. Furthermore, enriching the toolkit with additional support for (automated) verification and validation can serve as additional motivation to apply MBD-based tools for the demanding safety mechanisms.

Implications on Algorithms and Implementations

Given the AI background of MBD, it is evident that the state of the art tool-set utilizes advanced algorithmic and programming concepts. Our hybrid estimation scheme hME [Hofbaur and Williams, 2004], for example, utilizes advanced search with dynamic adaption to operational situations and on-line system analysis, decomposition and filter deduction. The Functional Safety standards, however, enforce low complexity and restrict the programming languages in terms of language constructs and data structures, see for example [mis, 2004], and even the tool sets such

as programming environments and compilers. As a consequence, we won't be able to implement hME within these restrictive constraints, nor would we be allowed to apply hME in safety critical environments (other MBD schemes will certainly experience similar difficulties).

High-fidelity MBD algorithms claim to master the complexity of the diagnosis task and usually provide good average run-time behavior. In the safety-critical context, however, we have to guarantee a worst-case run-time behaviour that is defined through the hazard analysis and risk assessment as introduced above. Providing any-time / any-space behaviour is thus an important step towards providing safety-qualified algorithms, but might not be enough.

4 Possible Solution Approaches for Safety-related Model-based Diagnosis

Given the discussion above, it becomes evident that MBD tools for safety critical applications need simple, well-trusted and approved/certified algorithms as basic tool sets that can be seamlessly integrated in the system's model-based control scheme.

For hybrid systems diagnosis, for example, we are currently reconsidering the classical IMM algorithm [Blom and Bar-Shalom, 1988] and enhance it with the gems of our high-fidelity syn-hME framework [Rienmüller *et al.*, 2013] – the simplified probabilistic modeling of mode transitions together with the ARR-based transition detection mechanism that provide simple and robust hybrid estimation.

For discrete models, there exist approaches for approximate compilation of the solution set into compact, graph-based representations such as OBDDs and DNNFs, enabling search-free on-line generation of the solutions [O'Sullivan and Provan, 2006]. Selective compilation of the solutions exceeding a particular preference threshold (for example, defined by probability or criticality) allows for further tailoring in order to meet given run-time and memory bounds. The simple enumeration of solutions from the compiled representations is more likely to fulfill traceability requirements than complex algorithms. However, this only applies to the on-line step and the whole process would thus still require certified compiler tools and verification of the models.

In the area of model-driven software development, there are approaches to capture the safety requirements – usually specified in natural language – in a more formal way, in order to (semi)automatically generate appropriate fault detection mechanisms that will fulfill these requirements in the system [Sojer *et al.*, 2011]. The interplay of diagnosis with formal safety requirements is an interesting direction that has not yet received much attention in MBD.

5 Conclusion and Outlook

With the increasing autonomy of technical systems in many industry sectors, such as automotive systems, the importance of Functional Safety to eliminate non-tolerable risks of injury or damage has been rapidly growing. We have given, from the point of view of model-based diagnosis, an overview of Functional Safety and one of its main contemporary industry standards. Rather than being a separate add-on, safety is addressed from the beginning of the system development process; starting from a risk assessment, the necessary safety goals and functions for risk reduction are derived and implemented as an integral part of the system's hardware and software functionality, together with its other functionality. For the model-based diagnosis research community to put its techniques to practice, we believe it will be mandatory to take into consideration Functional Safety norms and concepts. The international industry standards currently have low trust in on-line diagnosis methods, but this view can change if the MBD community succeeds to enrich and complement these methods with ways for easy verification, traceability, and certification of both the models and algorithms in order to comply with the strict requirements. Possible solutions directions, which we outlined in the paper, include a revival of simpler, but traceable and trustable algorithms, as well as the restriction to low-complexity but verifiable language and library constructs. Although this might in parts be perceived as a "step back", we argue that this holds a big promise from an application point of view, while at the same time it poses challenging problems for research.

References

[Blom and Bar-Shalom, 1988] H.A.P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33:780–783, 1988.

[Hofbaur and Williams, 2004] M. W. Hofbaur and B. C. Williams. Hybrid estimation of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 34(5):2178–2191, October 2004.

[iec, 2010] *Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems*. IEC 61508 International Standard, 2010.

[iso, 2011] *Road Vehicles – Functional Safety*. ISO 26262 International Standard, 2011.

[mis, 2004] *MISRA-C: Guidelines for the use of C language in critical systems*. MIRA Ltd, UK, 2004.

[O'Sullivan and Provan, 2006] Barry O'Sullivan and Gregory M. Provan. Approximate compilation for embedded model-based reasoning. In *AAAI*, 2006.

[Pfeufer *et al.*, 1999] R. Pfeufer, M. Müller, W. Haag, and F. Bederna. *Method and Arrangement for Monitoring the Detection of Measured Values in an Electronic Power control of a Motor of a Vehicle*. United States Patent 5.995.885, 1999.

[Rienmüller *et al.*, 2013] Theresa Rienmüller, Michael Hofbaur, Louise Travé-Massuyès, and Mehdi Bayoudh. Synergetic hybrid estimation. *Journal of applied mathematics and computer science (AMCS)*, 23(1):131–144, March 2013.

[Schäuffele and Zurawka, 2013] J. Schäuffele and Th. Zurawka. *Automotive Software Engineering*. Springer, 5 edition, 2013.

[Sojer *et al.*, 2011] Dominik Sojer, Christian Buckl, and Alois Knoll. Deriving fault-detection mechanisms from safety requirements. *Computer Science - Research and Development*, pages 1–14, 2011.

Fault Augmented Modelica Models

Johan de Kleer[1] and **Bill Janssen**[1] and **Daniel G. Bobrow**[1] and **Tolga Kurtoglu**[1]
Bhaskar Saha[1] and **Nicholas R. Moore**[2] and **Saravan Sutharshana**[2]
[1]Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
[2]Angeles Crest Engineering
2500 E. Foothill Boulevard, Suite 402
Pasadena CA 91107 USA
contact email: dekleer@parc.com

Abstract

Model-based approaches to diagnosis require behavioral models of components of the system to be diagnosed. While having models of correct behavior is sufficient to enable diagnosis, models of known faulty behaviors improves diagnostic precision. This paper assumes the existence of a correct behavioral model (at the component level) of a system, written in the popular modeling and simulation language, Modelica. We describe a semiautomatic way of extending the models of the system components to include modes in which these components can exhibit a large class of faulty behaviors. Some of these modes reflect changes in how the model transmits power. Others specify a faulty behavior based on physical changes derived from component use, such as frictional wear and corrosion. The latter behaviors are captured through parameter changes in the behavioral equations. This paper also describes how we construct probability distributions for the existence of these faults and the values of changed parameters.

1 Introduction

Availability of models of correct behavior is critical to the success of model-based approaches to diagnosis, repair, reconfiguration, etc. For many diagnostic algorithms [de Kleer and Williams, 1987; Reiter, 1987], the behavioral models are expressed in First-Order Predicate Calculus (FOPC) or one of its equivalents. Unfortunately, such models are hard to obtain from the engineers who designed the system. Modelica [Fritzson, 2004] has arisen, especially in Europe, as the modeling language of choice for Cyber-Physical systems. The open source Modelica Standard Library (MSL) includes models of approximately 1280 components in domains including electronics, mechanics and fluids. Hence, we are using these Modelica models more and more in our projects.

MSL describes nominal behavior, not faulty behavior. It is well known, that knowledge of fault modes greatly improves diagnostic precision [de Kleer and Williams, 1989]. This paper describes an approach to semi-automatically extending the MSL models to include such fault modes. Some of the possible fault modes can be determined from examination of the Modelica source code. Others arise from failure mechanisms. Failure mechanisms such as corrosion or wear lead eventually to failing components. For example, after a vehicle clutch has been used, it may no longer be able transfer enough torque to the wheels for the vehicle to move from a rest position. We provide models of common incremental failure mechanisms, and a means of computing how much wear is likely to result from specified use. Wear is represnted in a fault model by a change of a physical parameter (e.g. coefficient of friction). By contrast, fatigue failure in metals can result in a loss of function, which can be represented as a alternate behavioral mode.

We have constructed faultable Modelica models for almost all the component models in MSL. Each such model includes both its original nominal behavioral description and all of its fault modes. Thus for any system that is built from the nominal (unfaulted) component models, we can build a port to port equivalent system in which we can insert faults by substituting the faultable models for the originals.

The probability of faults or fault modes can vary by orders of magnitude. A brake is more likely to fail through wear than metal fatique. This paper also describes an approach to construct the probability density functions (pdfs) for faults. Consider a brake model. To model change in behavior due to wear we provide a function which provides the pdf of the remaining brake pad material after a certain age. In the brake case, age is represented by the cumulative distance the brake pads have slid against each other. This pdf is a function of the key parameters of a brake such as its geometry and material the brake pad is constructed from. We provide such a function for each augmented fault model. The pdfs may be non-parametric. Due to numerical complexity we represent these pdfs as tables and we compute exact values through interpolation.

These new fault models will support a designer in choosing components and their parameters so that their design will satisfy customer requirements. For example, suppose a clutch must be required to last at least two years of constant use. Unfortunately, the designer faces a circularity. To evaluate whether the clutch will meet this requirement one needs to have a fairly complete model of the vehicle, the type of terrain the vehicle will be driven on, and the types of missions it will be used for. We get ourselves out of this circularity as follows. First, we know that we are designing a vehicle. Given that, clutches, brakes, drive shafts, etc. are all going to be used in familiar ways. The damage mechanisms for all these parts are understood (simulations of damage with use are available to us as open-source Fortran codes). Therefore, using these we can build conditional probability distributions for each component as if it were generi-

cally used. For example, the probability density function for brake damage is conditioned on its geometry, type of material, normal force, etc. Terrain for vehicles is typically modeled as power spectral density functions (which concisely describes an infinite set of possible terrains with smooth roads, sand, boulders, etc.) Figure 1 illustrates the flow of information in constructing the fault augmented model along with its conditional density functions.

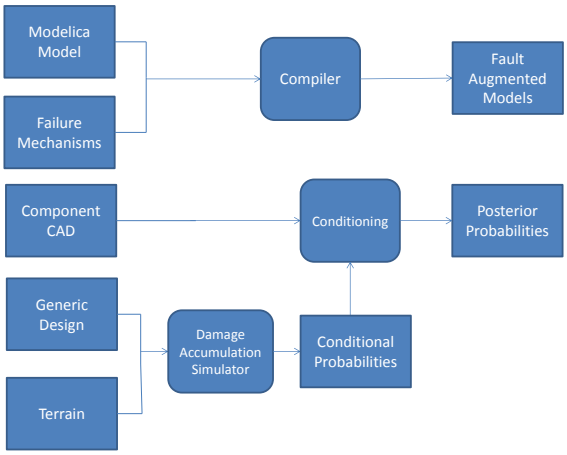


Figure 1: Basic FAME (Fault-Augmented Model Extension) archicture of our approach. We are given the base Modelica model, failure mechanisms, CAD information, and generic design of vehicle and terrain. There are three procedures in our framework: (1) the compiler constructs the fault augmented Modelica models, (2) the damage accumulation mechanism constructs conditional probability distribution and the dimenson reduction process conditions the density function by the particular component design (extracted from CAD information).

Many Modelica models are built out of more primitive components. We refer to Modelica models with no subcomponents (i.e., no internal *connect* statements) as leaf models. We obtain non-leaf fault models by inserting faults in the leaf models they include. In this way we are able to build faultable models for all entries in the Modelica Standard Library (and many more). We intend to make them available to the community as open-source.

2 Failure mechanisms

We consider the following failure mechanisms: corrosion, wear, metal fatigue, leaking, slipping, and open/short (in all their equivalents). These mechanisms include the majority of faults in the literature for cyber-physical systems such as vehicles.

Figure 2 is a simple Modelica model for an electrical capacitor.
We would like to be able to analyze it, see what faults it is susceptible to, and rewrite it so that these faults could be simulated in instances of this class.

In general, we look for three classes of faults: catastrophic, power flow, and parametric. Catastrophic faults typically change the dynamics of a component into something completely different; in the electrical domain, the most common example is an open circuit. Power flow faults affect

```
model Capacitor

  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  parameter Modelica.SIunits.Capacitance C;

equation
  i = C*der(v);
end Capacitor;
```

Figure 2: A Modelica model for a capacitor.

one or more of the power flows to or from the component; in electrical components, a common fault is a short circuit. Parametric faults typically reflect the shifting of a supposedly fixed parameter; in the case of a capacitor, the capacitance.

Once we have identified the faults a component is susceptible to, we then add behavioral descriptions for each of these faults to that component. Appendix 8 contains a version of the capacitor model with fault modes added (any more complex fault model are far larger than this and cannot be included in this paper). The model can now be run in any of these fault modes and the severity of each fault can be specified.

3 Augmenting Modelica Models

In the case of modeling fault modes through changes in the power flows, our approach is based on understanding the particular type of energy exchanged among components. This approach has the advantage of working with models specified quite abstractly, and in the face of modeling inconsistencies but the disadvantage of somewhat grossly modeling the actual failure behavior. In the case of modeling faulty behavior through parameter changes, models are augmented by using a manually compiled list of declarative specification of desired parametric faults, which are injected automatically. This allows us to model faults more specifically and precisely.

3.1 Fault analysis

Our power flow analysis relies on identification of standard power interfaces in the model. These are instances of Modelica connector classes, such as the `Pin` class in Modelica's package of analog electrical interface types, which contain two variables, one of an "effort" type, such as `Voltage` or `Pressure`, and another of a "flow" type, such as `Current` or `MassFlowRate`. The analysis examines both the directly defined components of the model class and components defined by inherited class that the model class extends.

In addition, our system can optionally read in a table of parametric faults, each row of which describes a particular fault in which a nominally fixed parameter changes during the operation of the component. For each fault, this provides the fault mode, the Modelica class, the specific parameter component of the class, the Modelica type of the parameter, and a Modelica expression describing the change in the parameter as a function of the variable `FAME_fault_amount`.

3.2 Model augmentation

Every model class definition which contains faults is replaced with a new class definition, a Modelica model class

subsuming the original model class and adding declarative behavior to allow simulation of the faults.

If a class model is found to be susceptible to one or more faults, the class is re-written. An encapsulated enumerated type is defined, listing the various fault modes of the class, along with the "Nominal" mode. A discrete mode parameter of this new type is defined, defining the mode in which an instance of the class is operating. An if-equation similar to Figure 3 is added, so that each operating mode can define its own dynamics.

```
if operating_mode == OperatingModes.Nominal then
   // original set of equations
   ...
elif operating_mode == OperatingModes.Fault1 then
   // equations for dynamics of Fault1 operation
   ...
else
   assert(False, "Invalid operating mode detected!");
end if;
```

Figure 3: Alternative dynamics are enabled for each operating mode.

The set of equations which apply in each fault mode is expressed in the appropriate branch of this if-equation. As the operating mode type is a Modelica `parameter`, the selected branch will not change during simulation and compilers can optimize this equation.

Each new class connects its instance of a power interface component through an added variable power dissipation component which in the nominal mode dissipates no power. For instance, if the original model class is an electrical component, and the power interface instance is an instance of the class `Pin`, the appropriate power dissipation component would be an instance of `FAME.DynamicDampers.Electrical`, with the `damping` parameter set to `0`.

In addition, if the shell class contains multiple power interfaces of the same type, it may also contain variable power conductance components connecting each pair of compatible connector components, nominally set to conduct no power. (Currently, this is only available for the `Electrical` domain.) Connector components are "compatible" if they are of the same type, or inherit from the same type. For instance, if the original model class is an electrical component which contains an instance "p" of the connector type `PositivePin` and an instance "n" of the connector type `NegativePin`, both of which are subtypes of `Pin`, there would be an instance of `FAME.Bridges.Electrical` connecting those two instances, with the `bridging` amount set to the very small value of `Modelica.Constants.eps`.

Appendix 8 contains an example of the Capacitor model with these dampers and bridges added. The process also flattens the superclasses of the model into the rewritten class, and introduces two new externally visible components, `FAME_operating_mode` and `FAME_fault_amount`, as well as an enumerated type giving the possible faults for this component, `FAME_OperatingModes`.

Faults which manifest as power flow anomalies can be modeled by a simple change to these two variables. For instance, as seen above, an electrical short can be modeled by setting `FAME_operating_mode` to `FAME_OperatingModes.Electrical_Short`, and `FAME_fault_amount` to 1.

Parametric faults are handled by introducing a new continuous variable, prefixed with `FAME__`. An equation is added to set this variable to the value computed by the function specified by the fault table. References to the original parameter are replaced with an expression which references to this new variable. For example, Appendix 8 replaces the `C` with `FAME__C = C*(1-FAME_fault_amount)`.

3.3 Implementation

Underlying Technology

[JModelica, 2013] is an open-source Modelica tool chain built by Modelon AB. JModelica consists of a Java/JastAdd parser for the Modelica language, along with various simulation and analysis tools, written in both Java and Python. For our purposes, only the Java/JastAdd parser is relevant. Version 1.9 of JModelica supports parsing of Modelica 3.2 language syntax, and is what our system is built with.

JastAdd [Hedin, 2011; Justadd, 2013] is a Java-based implementation of Knuth's *attribute grammars*, in which nodes in the abstract syntax tree (AST) can have *attributes*, the values of which are define by *equations*. Each attribute can be either *synthesized* (defined by an equation attached to the node itself), or *inherited* (defined by an equation in an ancestor node). JastAdd supports *reference* attributes, which is an attribute that has as its value another node in the AST. This allows arbitrary graphs to be "woven" through the AST. It also allows its attributes to be *parameterized*, which means that the *equation* has unbound variables which bound to parameters when the equation is evaluated. It also allows *collection* attributes, multi-valued attributes the value of which can be contributed to by various other AST nodes. And it supports *circular* reference attributes, which provide a way of breaking referential cycles.

JastAdd is Java-based, so *equations* are implemented as Java methods, and attribute access is via calls to those methods. It also extends Java itself with various constructs, notably *aspects*. JastAdd aspects support *intertype declarations* for AST classes. An intertype declaration is a declaration that appears in an aspect file, but that actually belongs to an AST class (like an attribute equation). The JastAdd system reads the aspect files and weaves the intertype declarations into the appropriate AST classes. It supports both *declarative* aspects (.jrag files), and *imperative* aspects (.jadd files). Declarative aspects add new attributes, equations, and rewrites; imperative aspects add only Java methods and variables. Rewrites of the AST can also be specified; they replace an AST node of type A with a node of type B, optionally only when some condition C is true. New AST subtrees can be created and specified as attribute values (*non-terminal attributes*).

Strategy

We use the JModelica parser to parse each library file, then examine each model class (non-partial Modelica class definitions with the restriction "model") found in the parse tree for instances of "power interface" connector classes.

If any of the above components are found, we create a new parse tree for a replacement class for the model class, incorporating the changes described above, and, using a JastAdd rewrite, replace the model class's parse tree with that new parse tree.

Finally, we use the JModelica "FormattedPrettyPrint" aspect to recreate the now-fault-enabled Modelica source code for the model from the AST.

The overall program for injecting faults into the models is therefore something like this (in pseudo-code):

```
for file in recurseOver (libraryTree):
    outputFile = figureOutputName (outputDir, file)
    parsedVersion = parseFile (file)
    prettyPrint (parsedVersion, outputFile)
```

Figure 4: Simple main loop for our fault injector.

Once the faults have been added, regular Modelica simulation can be used to assess their effects. Figure 5 is a plot of such a simulation of a brake model, both in nominal operating mode, and in both "sticking" and "slipping" fault modes.

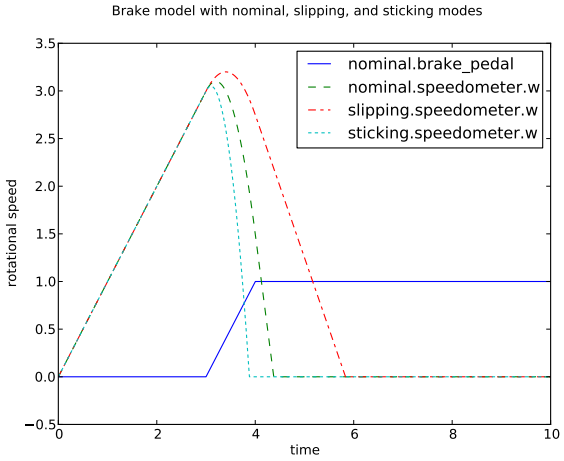


Figure 5: Plot of three simulations of a brake model fault with an attached speedometer. When the brake pedal is applied at time=3 seconds, the speed starts to slow. In the sticking fault mode, some drag is already present, and the rotational speed goes to zero more quickly. In the slipping fault mode, it takes more time to stop the shaft.

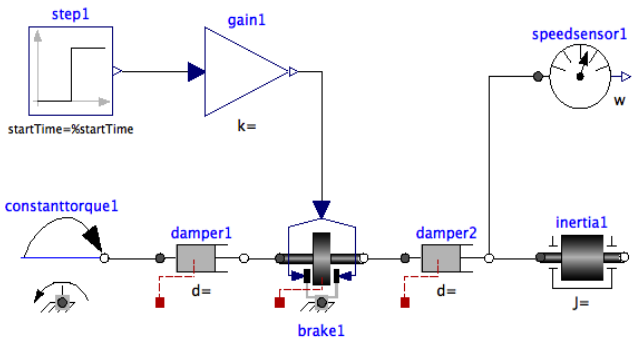


Figure 6: A simple use of the Faulted Augmented Modelica Brake.

4 Probability Density Functions for Damage

4.1 Surface Wear

We currently model the following parametric failure mechanisms: wear, fatigue, corrosion and stress rupture. In this paper we will only describe how wear and fatigue fits within our framework.

The most commonly accepted surface wear model is based on Archard's Law [Archard, 1953]. Archard's experiments with several different materials show that wear rate expressed as volume of material removed per sliding distance, *i.e.*, cubic inches per inch sliding distance, varies linearly with applied load and is independent of sliding velocity. In modeling brake wear for our simulations, the specific wear rate is assumed to be a constant and the friction coefficient is unchanged during a braking event. Using Archard's Law, the removed frictional material h is:

$$h = \frac{wFS}{A}$$

where:

w: Specific wear rate of friction material.

A: Area of frictional material.

F: Normal force pressing the surfaces together.

S: Sliding distance experienced by the brake (essentially its age).

Given a brake with a specific w, F, A and S we compute the wear fraction W_f as:

$$W_f = \frac{h}{h_{max}}$$

where h_{max} is the maximum depth of frictional material that can be worn away before friction becomes zero. In the slipping failure mode for the brake Modelica model the coefficient of friction parameter μ is replaced with $\mu(1 - W_f)$.

We model the random changes in the parameters w, A, F, S by scaling them with the factors $w^* \sim \mathcal{N}(1, 0.17)$, $F^* \sim \mathcal{U}(0.95, 1.05)$, $A^* = 1$ and $S^* \sim \mathcal{U}(0.9, 1.1)$ respectively (these distributions are derived from domain expertise). As a result the wear factor is a random variable denoted by $\tilde{W}_f$. Our goal is to precompute the probability density function of $\tilde{W}_f$ and store it with the model. Due to numerical complexity we can only compute an approximation of each by sampling. We compute these for a limited set of values of w, A, F, S that make engineering sense. At simulation time we use interpolation to determine the best $\tilde{W}_f$ given particular values of w, A, F, S Figure 7 illustrates one such pdf.

4.2 Fatigue

Materials that are subjected to cyclic loading may incur progressive damage that leads to failure even when the maximum load is well below the yield or ultimate strength of the material. Unlike wear, the damage parameter does not directly affect system performance, and when the component fails, failure is catastrophic. Number of cycles at failure has the form [Moore *et al.*, 1992]:

$$N = AS^{-m}e$$

S: strain in material (% change in length)

A: fit from data

m: fit from data

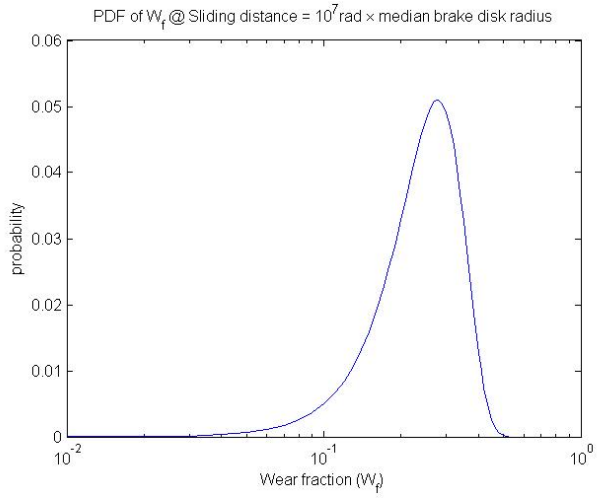


Figure 7: Brake W_f pdf for particular conditions.

e: drawn from a lognormal distribution constrained such that the median value of e for a given value of S and whose variance is estimated from least squares residuals.

The variable S is dependent on the geometry of the part and how it is used. Consider a gear. Life limiting fatigue damage of a gear can occur at the base of a tooth due to a torque load. The strain at a tooth base is given by:

$$S = \frac{T}{QE}$$

where
T : torque
Q : gear geometry factor
E : Young's modulus

Figure 8 is the resulting pdf for a particular gear using this approach.

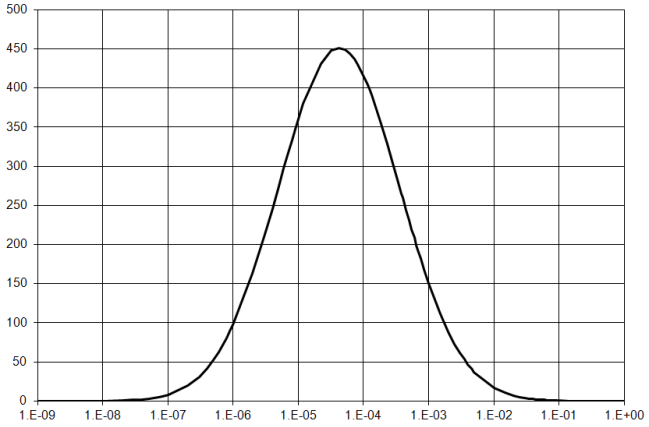


Figure 8: Gear damage probability density function.

5 Using fault augmented models

Fault augmented models can be used for a wide variety of diagnostic applications. All models are in standard Modelica. Thus, the initial values, parameters and fault modes can all be set in a Modelica wrapper. Some uses of the new models for both design and diagnostic purposes include:

- Given a set of initial conditions, parameter values and faults, determine whether a system requirement is met.
- Given a set of initial conditions, faults and component ages, determine the conditional probability that the system meets a requirement.
- Given a set of initial conditions, faults, observations, and ages for all but one of the components, determine the age of the remaining component.
- Perform standard model-based diagnosis.
- Perform fault mode and effects analysis (FMEA).
- Perform failure mode effects and criticality analysis (FMECA).
- Determine the sensitivity of meeting systems requirements with respect to fault conditions.
- Determine mean-time-to-faiure (MTTF).

We will briefly describe a few of these applications.

5.1 Evaluating whether a requirement has been met

Consider the simple braking system illustrated by Figure 6. Figure 5 describes the behavior of this brake for a particular set of initial conditions and parameters. Three operating modes of the brake are simulated: nominal, slipping and sticking. Suppose the requirement is that the vehicle must stop within 2 seconds of brake application, i.e. $P(T_{stop} < 2)$. Figure 5 shows what happens when the brake is applied at the 3 second mark. In the slipping case the angular velocity goes to zero at the 6 second mark, i.e., 3 seconds after the application of the brake. This simulation demonstrates that the system fails to meet its requirements.

5.2 Evaluating whether a requirement has been met given component ages

Continuing with this example, consider the case where the damage parameter is not known, but the age is known. Instead of a specific value, we can now use a probability density function of the damage parameter, as shown in Figure 7. We can sample from the pdf for brake wear fraction, W_f, and run multiple simulations to investigate the likely effect on stopping distance. Figure 9 illustrates a set of such simulations on the same graph. From the simulations we can see that the stopping time can vary by a few seconds. At the vehicle-level a change in stopping time by a few seconds translates to a significant change in stopping distance.

The system-level requirement, $P(T_{stop} < 2)$, can be evaluated over all the different simulations. In this particular case, the wear fraction pdf was representative of a brake used for 2000 hours. From the simulations, we compute the probability of not meeting the system requirement:

$$P(T_{stop} < 2) = 0.98$$

This means that there is a 2% chance of not meeting the 2 second braking time requirement.

5.3 Reliability Analysis at Component-level

The fault-augmented models together with component-specific damage pdfs will make it possible for designers to conduct reliability analysis on their system designs. In the simplest form, the failure probability of components may be derived from corresponding damage pdfs under a set of

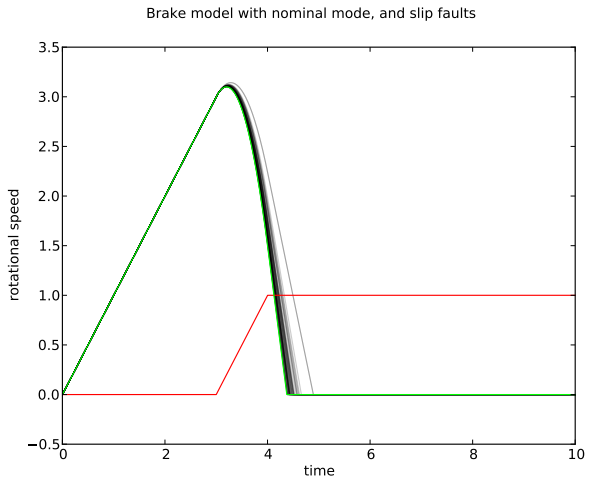


Figure 9: Multiple brake (Modelica) simulations.

pre-determined aging conditions, as described above. To determine the set of components to fault under the aging conditions, we can consider a ranked list of component failure probabilities or we can apply thresholds to the component-specific damage parameter distributions.

It is important to note that there can be different kinds of effect due to aging or any other degradation-causing mechanism. One type is the incremental degradation in model parameters, and another is the loss of function caused by aging or damage. Note that model parameters may be degraded even if the component is in the nominal mode. For the brake, friction wear causes smoothing of the friction surfaces, thus reducing the coefficient of friction between them (Figure 9). This is an example of incremental parametric degradation as discussed in section 5.2.

However, friction wear also causes material to wear away to the point that the friction surfaces do not make contact anymore. For the brake example, if the expected value of the brake wear fraction parameter, W_f, is more than, say, 0.6, i.e. $E(W_f) > 0.6$, then the brake may be considered to have been failed. Brake failure indicates that the component is no longer performing the braking function. This may be simulated by setting the `FAME_operating_mode` to `slip` and `FAME_fault_amount` to 1 for one of the dampers inside the brake model. The age at which this failure happens can be used to compute the brake life.

5.4 Reliability Analysis at System-level

We have built fault augmented versions of all models in the Modelica Standard Library. We are using them in many complex systems. One such system is shown in Figure 11, which is a Modelica model for a cross-drive transmission that is used in tracked vehicles to control both forward/reverse motion as well as steering. These systems can then be simulated under applicable operational scenarios with a set of non-nominal components and evaluated against a set of performance requirements. Based on the applicable damage-parameter distributions, the non-nominal components can have degraded model parameters or they may be set to simulate specific fault modes using the FAME damper or bridge elements.

Figure 10 illustrates the result of analyzing the cross-drive transmission in Figure 11 with fatigue failure in the left-side output gear. Fatigue failures in metals occur due to cyclic build up of strain as described in section 4.2. At the vehicle-level, this failure in the left-side output gear results in the vehicle being able to turn at roughly half its nominal rate. This approach is directly generalizable to any number of system parameters. The damage-parameter distributions of the system components can be sampled to run many such simulations. One approach is to use Monte-Carlo sampling to generate the simulation results. These results can then compared against system performance requirements to generate a probabilistic certificate of correctness (PCC) [Hoyle *et al.*, 2011].

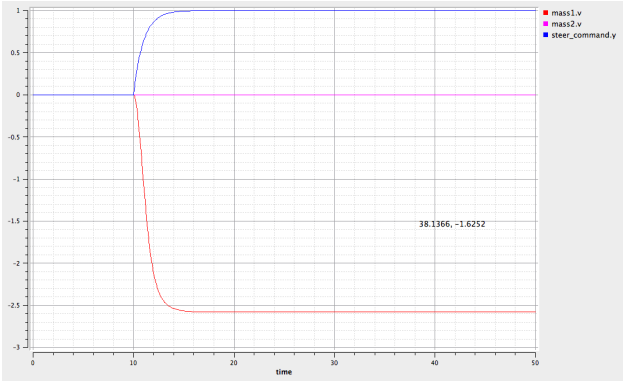


Figure 10: Modelica fault simulation of X-drive transmission. The top blue line is the command. The bottom line is the right track. The left track (center magenta line) does not move due to metal fatigue failure in left-side output gear.

5.5 Reliability Engineering

There are many approaches to describing the reliability of a system in the engineering literature (e.g., exponential, Rayleigh and Weibull). One of the most common frameworks is based on the exponential distribution where λ is the rate parameter. In this framework, the reliability of a system is defined as the probability of successful operation for a time t:

$$R(t) = e^{-\lambda t}.$$

The probability of system failure before time t is:

$$F(t) = 1 - R(t) = 1 - e^{-\lambda t}.$$

We can compute $\lambda(t)$ as follows:

$$\lambda(t) = \frac{\frac{dF(t)}{dt}}{1 - F(t)}.$$

$F(t)$ can be calculated directly from the conditional density functions described in Section 4. Hence, the $\lambda(t)$ required in formal analysis of hybrid systems can be determined directly from the conditional density functions.

5.6 Model-based Diagnosis

Most approaches to model-based diagnosis employ Bayes Rule. Let $\Theta = \{\theta_1, \ldots, \theta_n\}$ be the parameters of the system (e.g., resistances, capacitances and masses). Sequential diagnosis repeatedly applies

$$P(\Theta|o) = \frac{P(o|\Theta)P(\Theta)}{P(o)}.$$

for each new observation o. At the next iteration $P(\Theta)$ are the $P(\Theta|o)$ of the previous one. The damage pdfs provide the initial $P(\Theta)$. In model-based diagnosis the conditional probabilities $P(o|\Theta)$ are efficiently computed by drawing inferences upon the model of the system. In GDE $P(o|\Theta) = 1$ if o is consistent with prior observations, $P(o|\Theta) = 0$ is o is inconsistent with prior observations, and $P(o|\Theta) = \epsilon$ otherwise. This last case does not occur in our application because we use strong [Struss and Dressler, 1989] fault models. Which is fortunate in our case, as poor Modelica modeling practices result in many models being far more causal than one would like (one can include arbitrary imperative code in any Modelica model).

In many cases the probabilities can be represented in a simple closed form. Worst-case, a particle filter or MCMC is needed to perform the required computations.

6 Final Comments

Our approach is not as purely model-based as we would like. One avenue to improve our current approach would be to include the damage accumulation models inside the Modelica models themselves. Currently, we have to rely on a generic model written in Fortran. The validity of these models stems from their successful application in prior programs. Unfortunately, at least three impediments prevent a purely model-based approach: (1) Modelica has no ability to describe probabilities, (2) some of the Modelica models do not have any reference to certain fault and fault propagation mechanisms (for example, the brake model does not reference vibration which is conveyed to it via vehicle vibration caused by rough terrain), and (3) insufficient computational resources available to perform first-principles damage accumulation feedback immediately to the designer as he/she makes his/her design choices.

Our fault augmentation approach cannot model every conceivable fault. For example, if a resistor behaved as a capacitor (through some fault process or assembly failure), this cannot be modeled by a port failure or a parameter shift. We cannot model $v = iR$ changing to $i = C\frac{dv}{dt}$. Thus far in this project we have not encountered such a fault which occurs in practice.

We believe our fault augmentation approach can be extended to other modeling languages. The most commonly used language is Simulink. Unfortunately, Simulink is a primarily a causal language and Simulink models often cannot be automatically constructed from a description of the structure of a system. So it would not be very useful for model-based diagnosis.

```
model Capacitor

 import FAME;

// locally defined classes in Capacitor

 final encapsulated type FAME_OperatingModes =
    enumeration(Nominal, Drift, Electrical_Short,
                Electrical_Leak, Electrical_Break);

// components of Capacitor
 Modelica.SIunits.Voltage v;
 Modelica.Electrical.Analog.Interfaces.PositivePin p;
 FAME.DynamicDampers.ElectricalWithoutConnectEquations _damper_p;
 parameter Modelica.SIunits.Capacitance C;
 Modelica.Electrical.Analog.Interfaces.NegativePin n;
 FAME.DynamicDampers.ElectricalWithoutConnectEquations _damper_n;
 Modelica.SIunits.Current i "Current flowing from pin p to pin n";
 FAME.DynamicBridges.Electrical _bridge_p_n;
 parameter FAME_OperatingModes FAME_operating_mode=FAME_OperatingModes.Nominal;
 Modelica.Blocks.Interfaces.RealInput FAME_fault_amount;
protected
  Modelica.SIunits.Capacitance FAME__C;

// algorithms and equations of Capacitor
equation
 i = FAME__C*der(v);
 v = _damper_p.port_b.v-_damper_n.port_b.v;
 0 = _damper_p.port_b.i+_damper_n.port_b.i;
 i = _damper_p.port_b.i;
 connect(p,_damper_p.port_a);
 FAME__C = C*(1.0-FAME_fault_amount);
 connect(n,_damper_n.port_a);
 connect(p,_bridge_p_n.port_a);
 connect(_bridge_p_n.port_b,n);
 if FAME_operating_mode==FAME_OperatingModes.Nominal then
  FAME_fault_amount = 0.0;
 end if;
 if FAME_operating_mode==FAME_OperatingModes.Nominal then
  _damper_p.damping = 0.0;
  _damper_n.damping = 0.0;
  _bridge_p_n.bridging = 0.0;
 elseif FAME_operating_mode==FAME_OperatingModes.Electrical_Short then
  _damper_p.damping = 1.0;
  _damper_n.damping = 1.0;
  _bridge_p_n.bridging = 1.0;
 elseif FAME_operating_mode==FAME_OperatingModes.Electrical_Leak then
  _damper_p.damping = FAME_fault_amount;
  _damper_n.damping = FAME_fault_amount;
  _bridge_p_n.bridging = FAME_fault_amount;
 elseif FAME_operating_mode==FAME_OperatingModes.Electrical_Break then
  _damper_p.damping = 1.0;
  _damper_n.damping = 1.0;
  _bridge_p_n.bridging = 0.0;
 elseif FAME_operating_mode==FAME_OperatingModes.Drift then
  _damper_p.damping = 0.0;
  _damper_n.damping = 0.0;
  _bridge_p_n.bridging = 0.0;
 end if;
end Capacitor;
```

Figure 12: Our automatically generated capacitor model augmented with the ability to simulate several faults. Most augmented models are far more complicated.

7 Acknowledgments

This work was partially sponsored by The Defense Advanced Research Agency (DARPA) Tactical Technology Office (TTO) under the META program and is Approved for Public Release, Distribution Unlimited. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

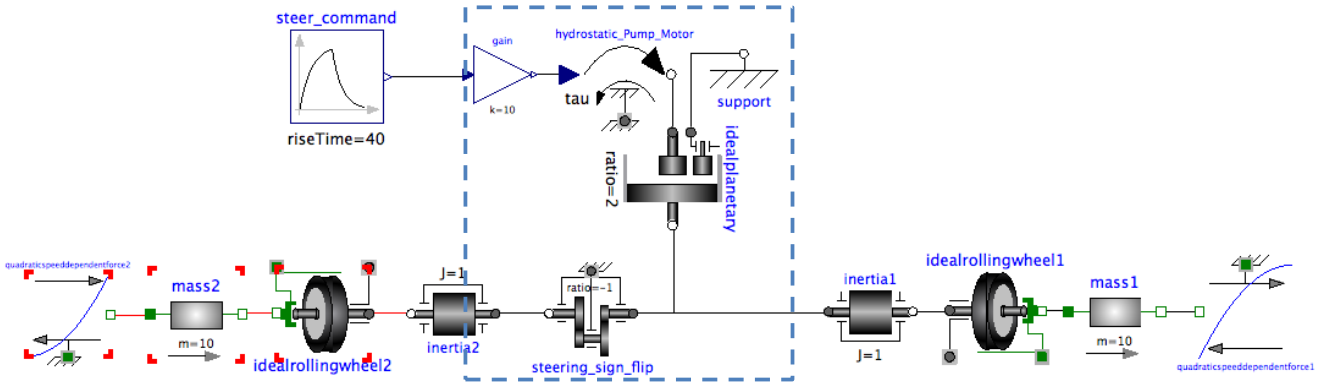


Figure 11: Modelica X-Drive transmission model.

8 Appendix

References

[Archard, 1953] J. F. Archard. Contact and rubbing of flat surfaces. *Journal of Applied Physics*, 24, 1953.

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

[de Kleer and Williams, 1989] J. de Kleer and B.C. Williams. Diagnosis with behavioral modes. In *Proc. 11th IJCAI*, pages 1324–1330, Detroit, 1989.

[Fritzson, 2004] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, Piscataway, NJ, 2004.

[Hedin, 2011] Görel Hedin. An introductory tutorial on jastadd attribute grammars. In João M. Fernandes, Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering III*, volume 6491 of *Lecture Notes in Computer Science*, pages 166–200. Springer Berlin Heidelberg, 2011.

[Hoyle *et al.*, 2011] Christopher Hoyle, Lin He, Irem Y. Tumer, and Tolga Kurtoglu. Multi-stage uncertainty quantification for verifying the correctness of complex system designs. In *Proceedings of the ASME 2011 IDETC and CIE*, Washington, DC, 2011.

[JModelica, 2013] The JModelica home page, `http://www.jmodelica.org/`, 2013.

[Justadd, 2013] Reference manual for JastAdd2 R20121026, 2013. See `http://jastadd.org/web/documentation/reference-manual.php`.

[Moore *et al.*, 1992] N.R. Moore, S. Sutharshana, and et al. An improved approach for flight readiness certification-probabilistic models for flaw propagation and turbine blade failure. JPL Publication 92-32, Vol. 1, Jet Propulsion Laboratory, California Institute of Technology, December 1992.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[Struss and Dressler, 1989] P. Struss and O. Dressler. Physical negation - integrating fault models into the general diagnostic engine. In *Proc. 11th IJCAI*, pages 1318–1323, Detroit, 1989.

8

Continuous State Estimation for Heterogeneous Hadoop Clusters

Shekhar Gupta[1], **Christian Fritz**[1], **Bob Price**[1], **Roger Hoover**[1], **Johan de Kleer**[1], and **Cees Witteveen**[2]
[1]Palo Alto Research Center, CA, USA
e-mail: {sgupta, cfritz, bprice, rhoover, dekleer}@parc.com
[2]Delft University of Technology, Delft, The Netherlands
e-mail: {c.witteveen}@tudelft.nl

Abstract

Hadoop is a popular and extremely successful framework for horizontally scalable distributed computing over large data sets based on the MapReduce framework. We present a monitoring tool for a heterogeneous Hadoop cluster to monitor real time performance of every node in the cluster. The performance of node measured in terms of slowdown. The monitoring tool is designed to help system administrators to detect underperforming node(s). Additionally, our tool also helps in identifying which resource (CPU or Disk) in the node is affected by the problem. In its current implementation, Hadoop assumes a homogeneous cluster of compute nodes. This assumption is manifest in Hadoop's scheduling algorithms, but is also crucial to existing approaches for detecting performance issues, which rely on the peer similarity between nodes. It is desirable to enable efficient use of Hadoop on heterogeneous clusters as well as on a virtual/cloud infrastructure, both of which violate the peer-similarity assumption. We have implemented the monitoring tool and present preliminary results on an eight node heterogeneous Hadoop cluster at PARC. We show that using our tool, resource specific performance problems (e.g., CPU contention, disk I/O contention) in a node can be detected by a system administrator.

1 Introduction

Hadoop[1] is a popular and extremely successful framework for horizontally scalable distributed processing of large data sets. It is an open-source implementation of the Google filesystem [Ghemawat *et al.*, 2003], called HDFS in Hadoop, and Google's MapReduce framework [Dean and Ghemawat, 2008]. HDFS is able to store tremendously large files across several machines and, using MapReduce, such files can be processed in a distributed fashion, moving the computation to the data, rather than the other way round. An increasing number of so called "big data" applications, including social network analysis, genome sequencing, or fraud detection in financial transaction data, require horizontally scalable solutions, and have demonstrated the limits of existing relational databases and SQL querying approaches.

Even though the value of Hadoop is widely acknowledged, the technology itself is still in its infancy. One of the problems that remains unsolved in the general case is the detection of faults and performance issues in the cluster. A Hadoop cluster may consists of hundred to thousands of nodes and there can be various kind of faults in any of the nodes. One can distinguish two types of faults, *hard faults* and *soft faults*. Crashing a node, disk failure, network failure can be seen as hard faults, resulting in failing jobs. Soft faults, on the other hand, consist in nodes still processing assigned tasks successfully, but at a lower than usual rate. Limping hardware, unnecessary background processes, or poor task scheduling resulting in overloaded nodes can all lead to soft faults. These faults create resource congestions such as CPU or I/O or network congestion, resulting in slowdowns. Hadoop uses a heartbeat protocol to detect hard faults. However, due to their dynamic behavior, detection of soft faults remains challenging.

Recently, [Tan *et al.*, 2010b] proposed a diagnosis approach that uses a simple *peer similarity* model to identify faulty nodes in a Hadoop cluster. The idea underlying their approach is that the same task should take approximately the same amount of time on each node in the cluster. Authors build statistical model of task completion time over the nodes of the cluster and identify outliers with fairly high accuracy. The approach relies on the assumption that the cluster is homogeneous i.e., that all nodes have the same hardware and software configuration. This assumption is not always true in real-world clusters where different machines may be purchased at different times.

To address the problems mentioned above, in this paper we propose a data-driven approach for detecting performance problems that is applicable to heterogeneous Hadoop clusters. This paper is an extension of our earlier work on diagnosis of heterogeneous Hadoop clusters [Gupta *et al.*, 2012]. In our previous work we studied an approach to detect resource specific soft faults in nodes. We presented a very simple probabilistic model to detect such soft faults. To build the probabilistic model we assumed that tasks resource requirements and node performance are already known. In this work the performance of a node is measured in terms of slowdown of a node. A slower node will process tasks at a lower rate compared to faster nodes. The approach was also implemented in an offline mode and had no notion of continuous monitoring. In this paper we extend the earlier work where we relax assumptions that requirements and performance metrics are known. Rather, we propose an unsupervised learning based approach to learn unknown parameters. To make our approach appealing we develop the monitoring tool which works in online mode, therefore soft-

[1] `http://hadoop.apache.org/`

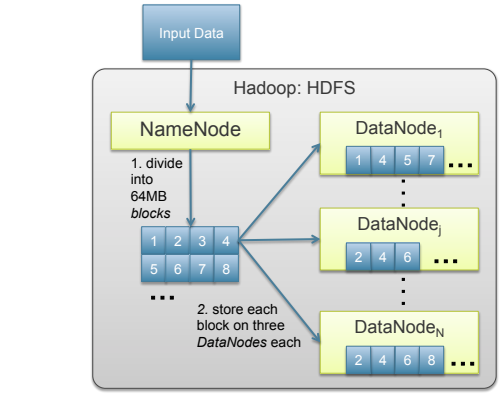
faults can be detected without stopping production. Our approach continuously estimates the slowdown of every node and the slowdown is measured for every kind of resource (CPU or Disk) separately. Hence, our solution is able to explain what kind of resources are affected in the presence of fault(s). While continuously monitoring the slowdown of nodes, radical increment in slowdown indicates the presence of a fault. This is accomplished without *any* additional input from the user or the cluster administrator.

The practicality of our solution relies on the structure of jobs in Hadoop. These jobs are subdivided into *tasks*, often numbering in the thousands, which are executed in parallel on different nodes. Mapping tasks belonging to different jobs can have very different resource requirements, while mapping tasks belonging to the same job are very similar. This is true for the large majority of practical mapping tasks, as Hadoop divides the data to be processed into evenly sized blocks. For a given node, we therefore monitor the performance of tasks running on the node in terms of task completion and infer the slowdown of nodes. Our inference engine depends on the tasks resource requirements to determine the nodes' resource performance. The inference engine is implemented as a probabilistic model. To build the probabilistic model we describe resource requirements and slowdown as random variable. We use the Gaussian distribution to model these random variables. To make our work more realistic we do not assume that the task resource requirements are known a priori. The resulting posterior distribution of the random variables is derived as an untractable four dimensional joint distribution. To avoid this problem we approximate this joint distribution in terms of marginals. The approximation is implemented as a two step iterative marginal estimations. In the first step, we estimate the expected resource requirements of tasks based on the current belief about the performance of nodes. In the second step, we update the belief about the slowdown of nodes based on the expected task requirements. Empirically we show that our iterative approach converges to meaningful values of node performance. We empirically demonstrate that our engine shows a real time performance of every node in the cluster. Using this real time monitoring tool underperforming nodes can be detected and the source of problem can also be identified.

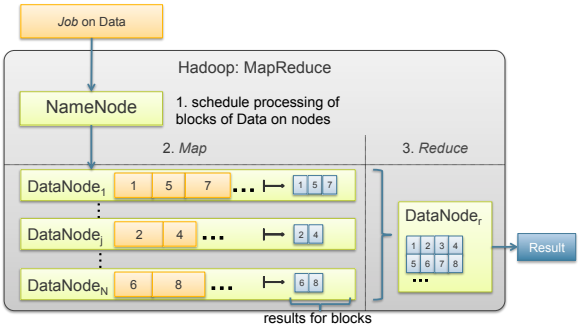
2 Hadoop Background

Hadoop is an open-source platform for distributed computing that currently is the de-facto standard for storing and analyzing very large amounts of data. Hadoop comprises a storage solution called HDFS, and a framework for the distributed execution of computational tasks called MapReduce. Figure 2 depicts how Hadoop stores and processes data. A Hadoop cluster consists of one *NameNode* and many *DataNodes* (tens to thousands). When a data file is copied into the system, it is divided up into *blocks* of 64MB. Each block is stored on three or more DataNodes depending on the replication policy of the deployed Hadoop cluster (Figure 1(a)). Once the data is loaded, computational *jobs* can be executed over this data. New jobs are submitted to the NameNode (Figure 1(b)). The NameNode will schedule *map* and *reduce tasks* onto the DataNodes.

- A map task processes one block and generate a result for this block which gets written back to HDFS. Hadoop will schedule one map task for each block of the data, and it will do so, generally speaking, by selecting one of the three DataNodes that is storing a copy of that block to avoid moving large amounts of data over the network.
- A reduce task takes all these intermediate results and combines them into one, final result that forms the output of the computation.



(a) HDFS



(b) MapReduce

Figure 1: HDFS stores files in blocks of 64MB, and replicates these blocks on three cluster nodes each. For a new *job* MapReduce then processes ("*maps*"') each block locally first, and then *reduces* all these partial results in a central location on the cluster to generate the end result.

The canonical example of a MapReduce program is `WordCount`, a program that counts the number of occurrences of each word in a large corpus of text. The Map function of `WordCount` tokenizes one block of the text, and counts words locally, line by line. The Reduce function would then take these local counts and sum them up to get the global result.[2]

Hadoop, as of the current version 0.22, does not take any performance differences between the DataNodes into account during the scheduling phase, but assumes a homogeneous cluster, i.e., servers that are equally fast in terms of CPU, disk I/O, RAM, and network bandwidth, etc.—the key parameters of task completion time. While Hadoop is particularly good for certain kinds of text analytics, in practice, various different kinds of jobs execute on a Hadoop cluster with different resource profiles. For each system resource, such as CPU, disk, RAM, and network I/O, there is a continuous spectrum of how much a given job makes use of it.

[2]For parsimony we are omitting certain details in this example to the extend of which they are irrelevant for this paper.

3 Related Work

Recently, [Tan *et al.*, 2010b] presented Kahuna, a diagnosis approach that uses a simple *peer similarity* model to identify faulty nodes in a Hadoop cluster. Roughly, the idea underlying their approach is that the same task should take approximately the same amount of time on each node in the cluster. More precisely, the authors build histograms of the time each task of a job takes on each machine. A node is identified as faulty when its histogram deviates from those of the other nodes. The authors show that this approach can detect slowdowns caused by various kinds of issues including CPU hogging, disk I/O hogging, as well as to a limited degree network package loss. The authors also show that different workloads have different "diagnostic power" in the sense that certain issues are not uncovered by certain jobs. This is consistent with our assumption of different job profiles. The authors do not describe whether Kahuna is able to detect what kind of fault may have occurred on a machine.

Kahuna assumes that the cluster is homogeneous, i.e., that tasks take roughly the same amount of time across machines. [Gupta *et al.*, 2012] empirically illustrates that, unsurprisingly, on heterogeneous clusters, the same task can take significantly longer or shorter depending on which machine is being used. During the experiment, we make sure that all machines were functioning flawlessly. Diagnosis hence cannot be based on the assumption that the same task should take equally long to execute on every node.

Many root cause analysis techniques use distributed monitoring tools that require active human intervention to locate the fault. Gangilia [Massie *et al.*, 2004] is a well known distributed monitoring system, which is capable of handling large clusters and grids. X-Trace [Fonseca *et al.*, 2007] and Pinpoint [Chen *et al.*, 2002] are tracing techniques to identify faults in distributed systems. Tan *et al.* [2010a] developed a visualization tool to aid humans in debugging performance related issues of Hadoop cluster. The tool uses the log analysis technique SALSA [Tan *et al.*, 2008] that uses the Hadoop log files and visualizes a state-machine based view of every nodes' behavior. Ganesha [Pan *et al.*, 2008] is another diagnosis technique for Hadoop, which locates faults in MapReduce systems by exploiting OS level metrics. Konwniski and Zahari [2008] use X-Trace to instrument Hadoop systems to investigate Hadoop's behavior under different situations and tune its performance.

Automated performance diagnosis in service based cloud infrastructures is also possible via the identification of components (software/hardware) that are involved in a specific query response [Zhang *et al.*, 2007]. The violation of a specified Service Level Agreement (SLA), i.e., expected response time, for one or more queries implies problems in one or more components involved in processing these queries. The methodology is widely accepted for many distributed systems. To a degree Hadoop is using this approach as well, as described above, but since the processing time for MapReduce jobs depend on many factors including the size of the data, only very crude limits can be used as cut-off. The determination of a reasonable cut-off is further hindered on heterogeneous clusters, where processing times can vary strongly between machines.

4 Approach

This section describes the methodology to implement the monitoring tool to detect resource specific performance anomalies in a node. Our monitoring approach is based on a model of task completion time that is defined in terms of *task requirements* and *server performance*. Unfortunately, these requirements and performance are not directly observable and there is no simple way to predict task execution times given only server hardware specifications and task source code. Instead, we adopt a learning based approach where node parameters are continuously learned from observed task completion times, using our model. In this work we only consider mapping tasks for the purpose of learning node parameters. We defer the modeling of and learning from reduce tasks to future work. In the remainder of this section, we introduce the task model and then describe our monitoring approach.

4.1 Task Model

The task performance model predicts the execution time of a task on a server given the task resource requirements and the performance of the server node. We model a task as a set of resource specific operation types such as reading data from disk/HDFS, performing computation, or transferring data over the network. The task resource requirements are represented by a vector $\theta = [\theta_1, \theta_2, \ldots, \theta_N]$ where each component represents resource requirement of a certain type (e.g., CPU, disk I/O, network I/O). These numbers are unit-less and will only be used relative to each other. Hence, it is not necessary to give meaning to these scalars in terms of, say, number of instructions, or bytes to read from disk. The performance of the server is measured by slowdown which is described by a corresponding vector $\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_N]$. The total time $T^{i,j}$ to process a task of job i on server j is the sum of the times of each resource requirement and the respective server performance.

$$T^{i,j} = \sum_k \theta_k^i \gamma_k^j + \Omega^j \tag{1}$$

Ω^j represents the fixed overhead to start the task on the server. We assume that every job imposes the same amount of overhead on a given machine. In this paper, we consider a two dimensional model in which $\gamma = [\gamma_c, \gamma_d]$ represents computation and disk I/O server performance and $\theta = [\theta_c, \theta_d]$ represents the corresponding task requirements. The task duration model will be:

$$T^{i,j} = \theta_c^i \gamma_c^j + \theta_d^i \gamma_d^j + \Omega^j. \tag{2}$$

In order to monitor the state of every node we use observations of map execution times $T^{i,j}$ to infer the values of γ_c and γ_d. For a specific node, a significant increase in γ_c or γ_d highlights a performance problem (node has become slower) such as a runaway process or disk contention. This will allow an administrator to identify the root cause of cluster performance problems.

4.2 Monitoring Module

We adopt a Bayesian perspective in which we start with a prior distribution over the parameters of the model $P(\gamma_c, \gamma_d, \theta_c, \theta_d)$ and update these using observations of mapping times $\{T^{i,j}\}_1^N$ where i is a task index and j is a server on which it was run to get the posterior $P(\gamma_c, \gamma_d, \theta_c, \theta_d | \{T^{i,j}\}_1^N)$.

We assume that the observed execution times $T^{i,j}$ are normally distributed around the value predicted by the task duration model of Eq. 2. The uncertainty is given by a standard deviation σ_j associated with machine j.

$$T^{i,j} \sim \mathcal{N}\left(\theta_c^i \gamma_c^j + \theta_d^i \gamma_d^j + \Omega^j,\ \sigma^j\right) \tag{3}$$

Given the likelihood function for observed time samples based on parameter values, the posterior distribution of the slowdown of a node and the resource profile of a job can be derived using Bayes rule. For our model with only CPU and disk resources, the likelihood has the form

$$p(T^{i,j} \mid \theta_c^i, \theta_d^i, \gamma_c^j, \gamma_d^j, \sigma^j) = \frac{1}{\sqrt{2\pi}\sigma^j} \cdot \exp \frac{\left(T^{i,j} - \theta_c^i\gamma_c^j - \theta_d^i\gamma_d^j - \Omega^j\right)^2}{2{\sigma^j}^2} \quad (4)$$

We do not know the exact functional form of the prior $P(\gamma_c, \gamma_d, \theta_c, \theta_d)$ nor the posterior $P(\gamma_c, \gamma_d, \theta_c, \theta_d | \{T^{i,j}\}_{i,j})$, but we do know that all of the variables are correlated by the observations. We propose an approximate decomposition in terms of the product of two bivariate normal distributions which we optimize using a heuristic re-estimation procedure. This approximation is motivated by the observation that the individual slowdown parameters γ_c and γ_d are linearly, negatively correlated given an observation because they enter into the likelihood linearly and represent a sum that explains the time taken. The intuition is that execution time can be explained by a fast CPU and slow disk or fast disk and slow CPU. We therefore employ a bivariate normal with full covariance matrix to represent the joint probability over the slowdown parameters. A similar argument motivates the use of a second bivariate normal to represent the symmetric linear correlation in the resource requirements parameters of the task.

$$P(\gamma_c, \gamma_d, \theta_c, \theta_d | \{T^{i,j}\}_{i,j}) = P(\gamma_c, \gamma_d | \{T^{i,j}\}_{i,j}) P(\theta_c, \theta_d | \{T^{i,j}\}_{i,j})$$

This factored joint is approximated by iteratively updating each component distribution using Algorithm 4.1. We initialize the slowdown parameters $\gamma_c^{j,t}, \gamma_d^{j,t}$ from offline experiments. The algorithm then computes the posterior distribution of the resource profile of every job in the cluster for the given time sample and prior values of node performance measure. In this step the mean values of the slowdown parameters, $\bar{\gamma}_c^{j,t}$ and $\bar{\gamma}_d^{j,t}$, are used to determine the task requirements $p(\theta_c^{i,t}, \theta_d^{i,t})$. Similarly in the second step, node slowdown parameters are updated based on the posterior distribution of the resource profile. The updated $p(\gamma_c^{j,t+1}, \gamma_d^{j,t+1})$ is used as the prior distribution for further iterations.

Algorithm 4.1: MONITORCLUSTER(T^{ij})

for each Iteration t ∈ TotalIterations

do $\begin{cases} p(\ \theta_c^{i,t}, \theta_d^{i,t} \mid \{\ T^{i,j}, \bar{\gamma}_c^{j,t}, \bar{\gamma}_d^{j,t}\ \}_j\ ) \\ p(\ \gamma_c^{j,t+1}, \gamma_d^{j,t+1} \mid \{\ T^{i,j}, \bar{\theta}_c^{i,t}, \bar{\theta}_d^{i,t}\ \}_i\ ) \end{cases}$

There are two major challenges to implement the proposed heuristic. First we need to derive the marginal distributions $p(\gamma_c, \gamma_d)$ and $p(\theta_c, \theta_d)$. Second, we need to explain how the performance parameters γ_c and γ_d are initialized.

4.3 Updating Marginals

In this section we derive the update for the marginal distribution of task requirements $p(\theta_c, \theta_d)$. The task model shown in Equation 2 is symmetric in terms of node slowdown and requirement parameters, so the same form of update can be used for the server performance parameters $p(\gamma_c, \gamma_d)$. To derive the posterior marginal distribution $p(\theta_c, \theta_d)$ we treat θ_c and θ_d as random variables in Equation 2. These random variable are assumed to follow bivariate gaussian distribution. The uncertainty about the slowdown is therefore captured by a covariance matrix $\Sigma_{\theta_c^i, \theta_d^i}$

$$[\theta_c^i, \theta_d^i] \sim \mathcal{N}(\ [\mu_{\theta_c^i}, \mu_{\theta_d^i}], \Sigma_{\theta_c^i, \theta_d^i}\ ) \quad (5)$$

We assume that the observed execution time $T^{i,j}$ is normally distributed as described in Equation 4. For the derivation, we substitute the expected values of $\bar{\gamma}_c^j$ and $\bar{\gamma}_d^j$ for the parameters. When a job is first submitted we assume that the resource requirements for its tasks are completely unknown. Assuming an uninformative prior, the posterior distribution after the first observation is just proportional to the likelihood.

$$p(\theta_c^i, \theta_d^i \mid T^{i,j}) = \frac{1}{\sqrt{2\pi}\sigma^j} \cdot \exp \frac{\left(T^{i,j} - \theta_c^i\bar{\gamma}_c^j - \theta_d^i\bar{\gamma}_d^j - \Omega^j\right)^2}{2}$$

For the second and subsequent updates we have a definite prior distribution and likelihood function. These two are multiplied to obtain the density of the second posterior update. Let the first experiment be on machine j with slowdown γ^j and let the observed time be T^j. Let the second experiment be on machine k with slowdown γ^k and let the observed time be T^k. The resulting posterior distribution is

$$p(\theta_c^i, \theta_d^i \mid T^{i,j}, T^{i,k}) = \frac{1}{\frac{1}{\sqrt{2\pi}}} \cdot \exp\left[\frac{\left(T^{i,j} - \theta_c^i\bar{\gamma}_c^j - \theta_d^i\bar{\gamma}_d^j - \Omega^j\right)^2}{2{\sigma^j}^2} + \frac{\left(T^{i,k} - \theta_c^i\bar{\gamma}_c^k - \theta_d^i\bar{\gamma}_d^k - \Omega^j\right)^2}{2\ {\sigma^k}^2}\right]$$

We omit the derivation for space, but we do give the update rules here. With every time sample we can recover the mean $\mu_{\theta_c^i, \theta_d^i}$ and covariance matrix $\Sigma_{\theta_c^i, \theta_d^i}$ by using the property of bivariate Gaussian distributions. Expanding the exponent of Equation 6 and collecting the θ_c^i and θ_d^i terms gives us a conic section in standard form:

$$a_{20}{\theta_c^i}^2 + a_{10}\theta_c^i + a_{11}\theta_c^i\theta_d^i + a_{01}\theta_d^i + a_{02}{\theta_d^i}^2 + a_{00} = 0$$

There is a transformation to map between the coefficients of a conic in standard form and the parameters of a Gaussian distribution. The mean and covariance of the distribution with the same elliptical form is given by:

$$\begin{bmatrix} \mu_{\theta_c^i} \\ \mu_{\theta_d^i} \end{bmatrix} = \begin{bmatrix} \frac{a_{11}a_{01} - 2a_{02}a_{10}}{4a_{20}a_{02} - a_{11}^2} \\ \frac{a_{11}a_{10} - 2a_{20}a_{01}}{4a_{20}a_{02} - a_{11}^2} \end{bmatrix} \quad (6)$$

$$\Sigma^{-1}_{\theta_c^i\theta_d^i} = \begin{bmatrix} a_{20} & \frac{1}{2}a_{11} \\ \frac{1}{2}a_{11} & a_{02} \end{bmatrix} \quad (7)$$

For every new time sample we compute coefficients a_{nm} of Equation 6. These coefficients determine the updated value of $\mu_{\theta_c^i}$, $\mu_{\theta_d^i}$ and $\Sigma^i_{\theta_c, \theta_d}$.

4.4 Initialize Node Slowdown Parameters

The initial values of server slowdown are estimated by executing probe jobs offline. Since the time we measure is the only dimension with fixed units, the value of the parameters is underdetermined. We determine the parameters of the system by choosing a 'unit' map task to define a baseline. The unit map task has an empty map function and it does not read or write from/to HDFS.

The compute and disk task requirements, θ_c and θ_d resp., are both zero, therefore Equation 2 allows us to estimate Ω. Multiple executions are averaged to create an accurate point estimate. Note that Ω includes some computation and disk I/O that occur during start up.

One could imagine attempting to isolate the remaining parameters in the same fashion, however, it is difficult to construct a job with zero computation or zero disk I/O. Instead we construct jobs with two different levels of resource usage defined by a fixed ratio η.

Let's assume we aim to determine γ_c. First we run a job $J_c^1 = \langle \theta_c, \epsilon_d \rangle$ with fixed disk requirement ϵ_d (J_c^1 might be a job which simply reads an input file and processes the text in the file). We compute the average execution time of this job on each server node. According to our task model the average mapping time for every machine i can be given as

$$T_1^i = \theta_c \gamma_c^i + \epsilon_d \gamma_d^i + \Omega^i \tag{8}$$

Next we run a job J_c^η which reads the same input but the processing is multiplied by η compared to J_c^1. Therefore, the resource requirements of J_c^η can be given as $J_c^\eta = \langle \eta\theta_c, \epsilon_d \rangle$. The average mapping time for every node can be given as

$$T_n^i = \eta\theta_c \gamma_c^i + \epsilon_d \gamma_d^i + \Omega^i \tag{9}$$

We solve for $\epsilon_d \gamma_d$ in Equations 8 and 9, set them equal, and solve for γ_c^i to get:

$$\gamma_c^i = \frac{T_n^i - T_1^i}{\theta_c(\eta - 1)} \tag{10}$$

This equation gives us γ_c^i in terms of a ratio. To make it absolute, we arbitrarily choose one node as the reference node. We set $\gamma_c^1 = 1$ and $\gamma_d^1 = 1$ and then solve Equation 10 for θ_c. Once we have the task requirements θ_c in terms of the base units for Server 1, we can use this job requirement to solve for the server slowdown on all the other nodes. Similarly we estimate γ_d. To avoid network communication while learning node slowdowns, we set the number of reducers to zero and set the replication factor to one. Table 1 gives an example of computed server slowdown parameters for an 8 node cluster of heterogenous machines.

Node	γ_c	γ_d	Ω
Node1	1	1	45
Node2	1	1	45
Node3	0.1	0.33	5.3
Node4	0.1	0.33	5.3
Node5	0.1	0.25	4.8
Node6	0.1	0.33	5.3
Node7	0.1	0.33	5.3
Node8	0.1	0.33	5.3

Table 1: Node Slowdown and Overhead

5 Experimental Evaluation

To evaluate the effectiveness of our monitoring tool we execute two MapReduce jobs `Pi` and `TestDFSIO` on the 8 node Hadoop cluster while continuously estimating γ_c and γ_d for every node. `Pi` calculates digits of Pi and starts 2000 mapping tasks on the cluster. `TestDFSIO` writes 4TB of files on the nodes of the cluster. The nodes in the cluster are heterogeneous in their hardware configuration since these machines were purchased at different times and for different original purposes. This heterogeneity is reflected in Table 1.

We conduct two experiments to demonstrate detection of CPU contention and Disk contention. First, we illustrate the nominal case. At the time zero, we start both the MapReduce jobs `Pi` and `TestDFSIO`. The green line shows the increase in CPU usage. The line increases slowly, as it takes a number of updates before the priors on the γ parameters are overcome by the data. At time 25:00, `TestDFSIO` completes and CPU slowdown falls.

To demonstrate detection of CPU contention, we repeat the experiment, but introduce a third CPU hogging task on Node 7 at time 10:00. Red lines in Figure 2(a), demonstrate that the monitoring system can detect the presence of of a CPU hogging job. The CPU saturates and does not show additional load until `TestDFSIO` completes at time 25:00. Unlike the nominal case, the CPU continues to be loaded by the hogging task, A similar effect can be seen for Disk I/O in Figure 2(b)), however the effect on DISK is smaller as the problem is due to CPU hogging, not disk I/O.

In Figure 3(a) and Figure 3(b) we can see that the monitoring system has more trouble isolating disk hogging processes. We start with the nominal behavior shown by the green line. Again, we start the `Pi` and `TestDFSIO` jobs. Once the `TestDFSIO` job completes at time 25:00, the CPU behavior returns to low levels.

To demonstrate the detection of disk hogging, we repeat the experiment, but introduce a disk hogging task on Node 3 at time 10:00. Again, due to resource saturation, we do not see an immediate effect. However, once `TestDFSIO` completes at time 25:00 we see that both the CPU and disk stay busy. While we can see that there is a fault present, we cannot clearly distinguish between a CPU and disk fault in this case.

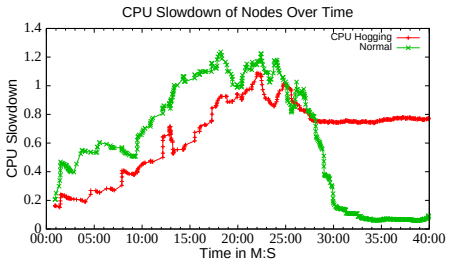
There was no impact of CPU or Disk hogging on the slowdown on nodes other than 7 or 3 for these experiments. Due to space constraint their results are not shown in the paper.

6 Future Work

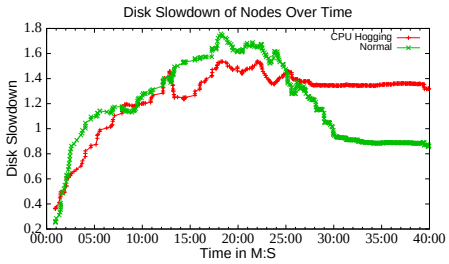
The monitoring tool described in this paper mainly assists a system administrator to detect lower performing resources in a heterogeneous Hadoop cluster. Therefore, we present slowdown nodes using plots and monitor the significant increment in slowdown. However, it would be more helpful if we can analyze the plotted data and automatically detect under performing resources. Since we are interested in significant changes in values for detection, an online change point detection [Adams,] approach would be a suitable choice for implementing an automatic detection engine. The continuous estimation of performance in terms of slowdown of resources can also be used to implement an optimized, dynamic task assignment policy building on our past work [Gupta *et al.*, 2013]. Intuitively, if, e.g., the CPU performance of a node goes down then fewer CPU intense tasks should be assigned to this node to maximize throughput.

7 Conclusions

We present a continuous state estimation approach for heterogeneous Hadoop clusters to detect performance issues on server nodes. The approach extends existing diagnosis approaches for Hadoop to clusters where the peer-similarity assumption does not hold, and further is able to distinguish between different types of faults. The approach does not require any specification of task requirements or server performance, but learns these parameters automatically by exploiting heterogeneity in the cluster. Using our monitoring tool a system administrator can not only discovers underperforming nodes but can also infer which resource in the node is lowering the performance. The paper presents a novel

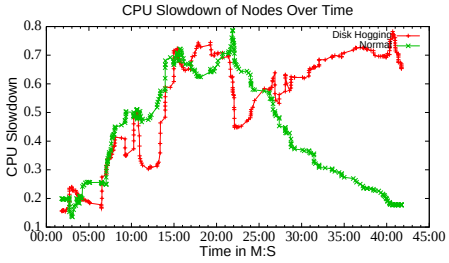


(a) Slowdown in CPU (γ_c) with CPU hogging

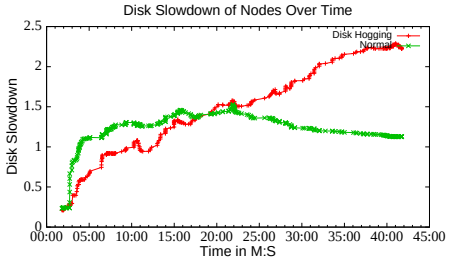


(b) Slowdown in Disk (γ_d) with CPU hogging

Figure 2: CPU hogging is injected in Node 7. γ_c and γ_d plotted under normal condition (Green) and CPU hogging (Red). Job `TestDFSIO` finishes at 25:00



(a) Slowdown in CPU (γ_c) with Disk hogging



(b) Slowdown in Disk (γ_d) with Disk hogging

Figure 3: Disk hogging is injected in Node 3. γ_c and γ_d plotted under normal condition (Green) and Disk hogging (Red).Job `TestDFSIO` finishes at 25:00

and simple iterative heuristic to approximate the joint distribution in terms of marginals. To empirically validate our diagnosis approach, we simulated soft faults in a Hadoop cluster consisting of 8 nodes, running two different MapReduce jobs with different resource profiles simultaneously. The preliminary results presented in this paper suggest that the proposed approach is viable and able to achieve the intended goals of a) identifying machines on which intermittent resource contention is occurring, and b) determining the resource which is over-subscribed by considering the relative impact of faults on the completion time for jobs with different resource requirements. While our demonstration uses the Hadoop system, the our approach is applicable to other frameworks of distributed computing as well.

References

[Adams,] Ryan Prescott Adams. Bayesian online changepoint detection.

[Chen *et al.*, 2002] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, O Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *In Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, pages 595–604, 2002.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[Fonseca *et al.*, 2007] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *In NSDI*, 2007.

[Ghemawat *et al.*, 2003] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system, 2003.

[Gupta *et al.*, 2012] Shekhar Gupta, Christian Fritz, Johan de Kleer, and Cees Witteveen. Diagnosing heterogeneous hadoop clusters. In *Proceedings of the 23rd International Workshop on Principles of Diagnosis (DX)*, 2012.

[Gupta *et al.*, 2013] S. Gupta, C. Fritz, R. Price, R. Hoover, J. de Kleer, and C. Witteveen. Throughputscheduler: learning to schedule on heterogeneous hadoop clusters. In *International Conference on Autonomic Computing (ICAC '13), June 26-28, 2013, San Jose, CA USA*, 2013. To appear.

[Konwniski and Zahari, 2008] Andy Konwniski and Matei Zahari. Finding the elephent in the data center: Tracing hadoop. Technical report, 2008.

[Massie *et al.*, 2004] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

[Pan *et al.*, 2008] Xinghao Pan, Jiaqi Tan, Soila Kavulya, Rajeev G, and Priya Narasimhan. Ganesha: Black-box fault diagnosis for mapreduce systems. Technical report, 2008.

[Tan *et al.*, 2008] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Salsa: Analyzing logs as state machines. In *First USENIX Workshop on the Analysis of System Logs (WASL), Dec. 7, 2008, San Diego, CA, USA*, 2008.

[Tan *et al.*, 2010a] Jiaqi Tan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Visual, log-based causal tracing for performance debugging of mapreduce systems. In *ICDCS*, pages 795–806, 2010.

[Tan *et al.*, 2010b] Jiaqi Tan, Xinghao Pan, Eugene Marinelli, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Kahuna: Problem diagnosis for mareduce-based cloud computing environments. In *IEEE/IFIP Network Operations and Management Symposium*, 2010.

[Zhang *et al.*, 2007] Rui Zhang, Steve Moyle, Steve McKeever, and Alan Bivens. Performance problem localization in self-healing, service-oriented systems using bayesian networks. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07. ACM, 2007.

Automated Functional Safety Analysis of Vehicles Based on Qualitative Behavior Models and Spatial Representations

Peter Struss, Sonila Dobi
Tech. Univ. of Munich
{struss, dobi}@in.tum.de

Abstract

For automotive vehicles, the analysis of functional safety has been strongly enforced in recent years and is subject to international standards. This paper presents an application of qualitative model-based and spatial reasoning with the aim of automating a major part of the process. The problem and model is split into two parts: first, a qualitative model of the vehicle subsystem (the drive train of a truck in our case study) is used to predict the effect of a component fault on the behavior of the entire vehicle, such as an unintended acceleration. Secondly, the impact of this effect on the environment of the vehicle has to be determined, e.g. a collision of the vehicle with persons or objects. This requires a model of the environment and the interaction of the vehicle with it and, hence, a spatial representation of positions of the vehicle and other objects relative to the road and their interference under different scenarios.

1 Introduction

Analyzing whether a technical system performs safely even under the occurrence of a fault is of high importance when its failure may cause injury or death of humans or other severe damage to its environment. For automotive vehicles, safety analysis has been strongly enforced in recent years and is subject to international standards. This task, which may have to be carried out repetitively for different versions and variants during the design of a system, is knowledge-intensive and consumes significant efforts of experts. Currently, there are no tools supporting and automating the reasoning part of this task. Achieving this through knowledge-based systems solutions that reduce the labor cost and improve the guaranteed coverage and quality of the results is of high importance under economic, social, and environmental aspects.

We present the realization and evaluation of a prototypical solution to the problem in a case study on the drive train of a truck ([Dobi et al., 2013]). It was carried out together with an industrial partner, who provided the subject, the requirements, and the evaluation criteria (in terms of a manually generated safety analysis).

In our solution, we exploit previous research results on qualitative deviation models ([Struss, 2004]) and automated model-based FMEA ([Price, 2000], [Picardi et al., 2004]). Beyond this, it provides several **scientific contributions** and **novel solutions**:

- A **conceptualization** of and a **systematic approach** to the task of **functional safety** analysis of cyber-physical systems, which does not exist in the literature, so far.
- An application of **qualitative deviation models** to a class of **mechanical systems** combining torques and forces and their control unit software.
- The development of a **spatial representation** of the motion of road vehicles and its environment as the basis for the automated analysis of the impact of a component fault on safety.

The following section describes the application context of the task and the drive train case study. Section 3 discusses the approach to functional safety analysis and its formalization. Modeling the drive train and inferring **hazards**, i.e. abnormal **behavior** of the **vehicle** caused by component **faults** are presented in section 4, while the following section describes how to determine the **impact** of an **abnormal motion** of the vehicle on its **environment**.

2 The Task

2.1 Safety Analysis in the Automotive Industries

The number of accidents, casualties, and injuries caused by automotive vehicles, but also other kinds of impact on the environment, e.g. through pollution, is a big concern and has led to many technical solutions (from anti-lock braking systems to sophisticated driver assistance systems), legal regulations (e.g. OBD2), practices and processes (Failure-modes-and-effects and criticality analysis, FMECA, Fault-tree analysis, FTA), and standards (e.g. IEC 61508).

Through a recent standard, ISO 26262 on Road Vehicle Functional Safety focusing on E/E (electrical and electronic) systems [ISO-26262, 2011], the necessity to carry out thorough and vast analyses of vehicle safety and steps towards preventing unacceptable risks caused by system design or component failure has been greatly emphasized.

In the analysis phase, the causal relationships between faults occurring in the system and hazards, i.e. unintended behavior bearing the risk of damage, has to be determined, as well as scenarios under which this damage may occur, its severity, and whether it can be controlled by the driver. If unacceptable risks are not excluded, effective policies have to be introduced into the design (e.g. in terms of structural

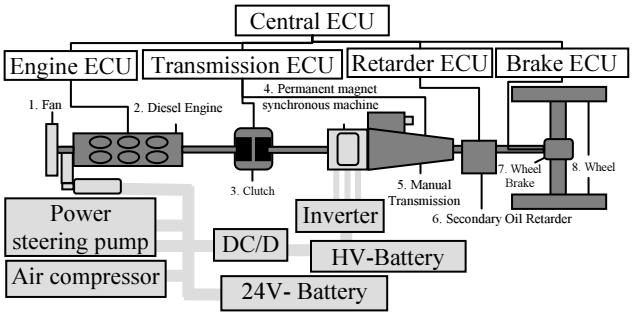


Figure 1. Drive Train Model

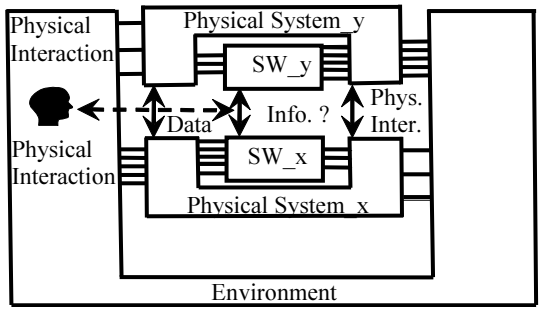


Figure 2. Cyber-physical Systems

changes and redundancy, additional sensors, or modified software functions).

2.2 The Drive Train Case Study

Our industrial partner selected a drive train of a truck as the subject of a case study. Its structure is sketched in Figure 1. The main part (in dark gray) comprises the engine, which produces torque for acceleration, but also for braking, the clutch, which may interrupt the propagation of torque, the transmission allowing to switch between forward and reverse torque (and idling), the retarder, a braking device that, when applied, counteracts the rotational motion through a propeller moving in oil, and the axle with the wheel, which transforms rotational acceleration into translational acceleration (and vice versa), and the wheel brakes. Components are controlled by specialized Electronic Control Units (ECU), which communicate with a central ECU that processes, for instance. the driver demands. The light-gray components are related to electrical aspects and are not discussed in this paper.

The industrial partner also supplied us with documents on exemplary problems and manually generated safety analysis tables. The core of an entry in such a table links a component fault (e.g. "erroneous CLOSE command to the clutch"), a special driving situation ("engine running, vehicle standing"), and a type of scenario ("vehicle in front of pedestrian crossing") with a hazard ("unintended forward acceleration") and its impact on the environment ("injury of persons"). Relevant impacts are typically hitting objects or persons, where, obviously, the severity is influenced by the type of object. More details are provided in [Dobi et al., 2013].

3 Safety of Cyber-Physical Systems

As mentioned before, there is no lack of standards and current practices. However, they do not provide a formal foundation for a computer-based solution. Hence, a systematic and structured approach to functional safety analysis of systems with embedded software had to be developed and mapped to formalized solutions in model-based problems solving. We present the solution using our case study as an illustration. Its background is illustrated by Figure 2: a cyber-physical system (CPS) comprises a number of subsystems, which are systems composed of physical (mechanical, electrical, hydraulic, etc.) components and software components, whose interaction happens exclusively through a usually relatively small set of sensor signals as the input to the software components and actuator signals as their output. Different subsystems interact both via connections between their physical components and via communication between their software components. In a vehicle, the components of the drive train with their individual ECUs are examples for such subsystems. At a higher level, the drive train itself can be considered as a subsystem. The top level system is the entire vehicle.

From the perspective of safety analysis, it is important to note that it is **only** the vehicle as a **physical** system that interacts with the environment. The embedded software never directly interferes with the environment. As a consequence, **hazards**, misbehaviors that bear the potential of damage in the environment, are defined **exclusively** at the intersection of the **physical system** and the **physical** environment, in our example, an unintended motion of the vehicle relative to its environment. As an important consequence, buggy software behavior matters if and only if it may cause the physical system to create a hazard.

In turn, hazards create risks only through their impact on the environment. Obviously, this environment is much more diversified and dynamically changing compared to the designed artifact, the vehicle. It cannot be explored exhaustively, but only through certain abstract types of scenarios and driving situations as illustrated by the example mentioned in 2.2.

In consequence, we approach the task of building a tool for safety analysis by dividing it (conceptually) into two steps (Figure 3):

- **hazard analysis**: a **behavior model** of the CPS (i.e. the relevant subsystems of the vehicle) is used to determine whether assumed faults of (software or physical) components may result in (pre-defined) hazards for a set of specified scenarios, in our case **driving situations** (in terms of speed, driver actions, etc.) and **road conditions** (slope and surface friction),
- **impact analysis**: a **model of the environment**, relating positions and motions of the vehicle and other objects and agents, determines whether the fault/hazard may have a dangerous **impact** (in our case, a collision) under certain **environmental conditions**, in the example specified by the driving

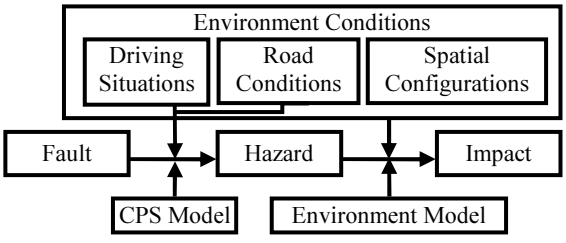


Figure 3. Hazard and Impact Analysis and their Inputs

situation, road conditions (including curvature) and the spatial configuration of other objects.

The two models together associate component faults directly with safety violations and risks.

For characterizing and implementing the required inferences, hazard analysis can be formalized as the task of determining whether a hazard HAZ_k may occur under a model of the system with an assumed fault, $MODEL_f$, in a scenario $SCEN_j$:

$$MODEL_f \cup SCEN_j \cup HAZ_k \not\models \bot,$$

or, stronger, is entailed by them:

$$MODEL_f \cup SCEN_j \models HAZ_k.$$

Hazard analysis is basically identical to failure-modes-and-effects analysis (FMEA) (where hazards correspond to effects) and iterates over the Cartesian product of scenarios and faults and performs the above checks for each defined hazard. Under qualitative abstraction, each of the three elements can be represented as a set of constraints or, if preferred, as first order formulas, and the analysis can be carried out using a constraint solver. In our case study, we used the FMEA engine of Raz'r [Raz'r, 2013]. Under a different view, a hazard is associated with a fault, if $MODEL_f$ is a consistency-based diagnosis of the (hypothetical) observations $SCEN_j \cup HAZ_k$.

Impact analysis is formalized as checking whether $MODEL_f$, the environment model, $MODEL_{env}$, and a scenario, $SCEN_{env,j}$, are consistent with an impact, $IMPACT_l$:

$$MODEL_f \cup MODEL_{env} \cup SCEN_{env,j} \cup IMPACT_l \not\models \bot,$$

where $SCEN_{env,j}$ is usually an extension of $SCEN_j$ by environmental conditions.

If hazards are determined as an intermediate result, the criterion becomes

$$HAZ_k \cup MODEL_{env} \cup SCEN_{env,j} \cup IMPACT_l \not\models \bot.$$

In either case, it is apparent that impact analysis can be implemented using the FMEA engine, as well.

In order to understand that the consistency criterion introduced above is not too weak, one has to consider that we formalize and implement an inherently qualitative and worst-case analysis. Firstly, the analysis is performed at design time, and parameters may not yet have numerical values. Beyond this, the faults are qualitative: decreased friction of a brake, a leakage of a pipe, and a high sensor signal cannot be described by numerical values. Hazards are qualitative: too high or too low acceleration are not specified more precisely than in this qualitative way. Scenarios are qualitative: "a vehicle approaching a pedestrian crossing with medium speed" or "going downhill a winding road". With regard to the required inferences, the worst-case analysis is not expected (and, given the qualitative input, not able) to firmly conclude the impact. What needs to be determined is the **potential** of a collision, e.g. given a reduced deceleration of the vehicle and, hence, a longer brake path – after all, for the instances of the pedestrian crossing scenario, it is not even clear whether there will be any pedestrians present. Finally, determining that a brake with reduced friction causes a reduced deceleration suffices to consider it as a reason for a risk in the respective scenario.

4 Model-based Hazard Analysis

In the following subsections, the elements needed for hazard analysis are described for the drive train case study:

- a **model** of the respective **system** (physical and software components), in which models of the component fault can be injected,
- a definition of relevant **driving situations and road conditions**, for which the analysis has to be carried out,
- a definition of the relevant **hazards**,

and we discuss the results obtained.

4.1 Drive Train Model

Reflecting the basic requirements of the analysis task, the models we developed are

- **behavior models** (as opposed to functional models as proposed e.g. in [Kurtoglu-Tumer-Jensen, 2010]), because the analysis needs to be based on determining the system performance objectively (based on 1st principles models), and completely, (not reduced by the expected function of intentions of the designers). Those aspects are captured by the scenarios and hazards considered,
- **qualitative models**, according to the nature of the analysis, as discussed in the previous section,
- **deviation models**, since faults, faulty behaviors, and hazards express (significant) deviations from a reference (nominal or intended state),
- **component-oriented, compositional models**, firstly because the analysis is highly repetitive, in having to be performed several times during the design phase, applied to alternative designs, and subsequently to different versions and variants, and a model-based solution is economically beneficial only if building the model is cheap. Secondly, components are the building blocks of the system and also the entities introducing the faults, and the same holds for their models. Note that the qualitative level of modeling increases the re-usability of models, because many intricate, irrelevant distinctions are abstracted away.

The components of the drive train determine the **acceleration or deceleration** of the vehicle.

These considerations indicate that the modeling task is non-trivial. The issues to be addressed are

- The overall (deviation of the) torque applied cannot be determined locally, but only as the **combined impact** of several components.
- The transformation of **torque** into an **accelerating force** and vice versa
- The modeling of software components and, especially, **software faults**, which seems to be in the complexity class of clearing out the Augean stables.

We discuss these aspects in the following.

Deviation Models

We use deviation models ([Struss, 2004]) in the same way as in a case study of FMEA of a braking system [Struss-Fraracci, 2012]: the qualitative deviation of a variable x is defined as

$$\Delta x := \text{sign}\,(x_{act} - x_{nom})$$

which captures whether an actual (observed, assumed, or inferred) value is greater, less or equal to the nominal value. The latter is the value to be expected under nominal behavior, technically: the value implied by the model in which all components are in OK mode.

Faults may introduce non-zero deviations, e.g. the model of a worn brake would result in a deviating braking torque, which depends on the direction of the rotation (kinetic friction)

$\Delta T_{brake} = \omega$

or the applied torque in case of static friction

$\Delta T_{brake} = T_{wheel}$

Models of OK and faulty behavior are stated in terms of constraints on the deviations. For instance, a correctly closed clutch simply propagates a deviating torque coming on the left from the engine to the right (flipping the sign):

$\Delta T_{right} = -\Delta T_{left}$.

Most models variables and all deviations have values from the domain Sign = {-, 0, +}: torques and forces, T and F, rotational and translational speeds, ω and v. The commands and states explicitly discussed here have Boolean values {0, 1}.

In the following, we outline the key ideas and illustrate them by selected component models. The models were manually created. How to automate their generation from existing numerical models is important, but not subject to this paper (see e.g. our previous work in [Struss-Fraracci-Nyga, 2011]).

Drive Train Modeling: Combining Torques

The core purpose of the drive train component models is to determine the **combined** (deviations of the) **torques** of the various components acting on the axle, the transformation of **forces and torques** into each other through the interaction of the wheel with the road surface dependent on friction, and the relation to translational **acceleration** and speed of the entire vehicle. Things get even more complicated, when the road has a non-zero slope and **gravity** adds a force that accelerates (or decelerates) the vehicle – again, dependent on friction: with sufficient friction, the gravity component along the road will add another torque to the axle (which may be overcome by other torques), otherwise, it will directly contribute to the translational acceleration of the vehicle (sliding downhill). In this paper, we ignore the impact of slope and friction, in assuming that friction suffices to strictly link torque and force and that the road is level.

The overall torque results from the interaction of all components, which potentially contribute to it. The engine can produce a driving torque, the braking elements (wheel brake, retarder, and engine) may generate a torque opposite to the rotation, and the clutch and transmission may interrupt or reverse the propagated torque.

Our current model is based on assuming that there are no cyclic structures among the mechanically connected components, which is the case in our application, but certainly also in a much broader class of systems. The component models link the torque (deviation) on the right-hand side to the one on the left-hand side, possibly adding a torque (deviation) generated by the respective component. Hence, at each location in the drive-train model, the torque (deviations) represent the sum of all torques collected on its left-hand side.

Whenever a terminal component (in our case the wheel) or a component in an open state (the clutch and the transmission) is reached, the arriving torque is the total one for the section left, and for the open components, the torque on the right-hand side is zero, as exemplified by the clutch (state=0 means open):

$state=1 \Rightarrow T_{right} = T_{left}$

$state=0 \Rightarrow T_{total} = T_{left} \wedge T_{right} = 0$.

Determining the deviation models is not as straightforward, as it may appear, as we will explain using the model of the retarder as an example. If engaged (state=1), it will generate a torque opposite to the rotation (zero, if there is no rotation) and add it to the left-hand one. The base model is obvious:

$T_{right} = T_{left} \oplus T_{brake}$

$state = 1 \Rightarrow T_{brake} = -\omega$

$state = 0 \Rightarrow T_{brake} = 0$,

where $\oplus$ denotes addition of signs. The first line directly translates into a constraint on the deviations:

$\Delta T_{right} = \Delta T_{left} \oplus \Delta T_{brake}$

However, determining ΔT_{brake} requires consideration of how the actual state is related to the nominal one, which depends on the control command to the component, and, to complicate matters, not on the actual command, but the **command that corresponds to the nominal situation**. This means we have to model possibly deviating commands, and we apply the concept and even the definition of a deviation also to Boolean variables. For instance, in the retarder model, $\Delta state$ = - means $state = 0$ (i.e. it is not engaged) although it should be 1, and $\Delta state$ = + expresses that it is erroneously engaged. Such deviations could be caused by retarder faults, e.g. stuck-engaged. However, in the context of our analysis, we must consider the possibility that the commands to the retarder are not the nominal ones (caused by a software fault or the response of the correct software to a deviating sensor value). Under multiple faults, a component fault may even mask the effect of a wrong command (the retarder stuck engaged compensates for Δcmd = -). In the OK model of the retarder, the actual state corresponds to the command, and the deviations of the command and state (i.e. the real, physical state) are identical:

$\Delta state = \Delta cmd$.

For a stuck engaged fault, however, Table 1 captures the constraint on the deviations:

Table 1. Retarder stuck engaged - Deviation constraint

cmd	Δcmd	Δstate
1	0	0
0	0	+
0	-	0
1	+	+

Here, the third row represents the masking case mentioned above, the first one reflects that the physical state coincides with the command, while in the second one, it does not.

From $\Delta state$, ΔT_{brake} is determined by

$\Delta T_{brake} = -\omega \otimes \Delta state$,

where $\otimes$ denotes multiplication of signs. This completes the model of the retarder.

Software Models

Since the drive train contains a number of ECUs, we also need to include models of software **and its faults** in our library. Remember: all that matters about software faults is their impact on the physical system, more precisely, on the controlled actuators. For the Boolean commands in our model, this means the only fault types to be considered are

- **Missing** (or late) **command**: $\Delta cmd = -$
- **Untimely** (or early) **command**: $\Delta cmd = +$.

The same applies to continuous actuator signals, where the faults represent signal too low and too high, respectively.

This provides evidence for the claim that putting safety analysis back on its feet and the physical model in the center, greatly simplifies the modeling and analysis of the embedded software. In particular, for the purpose of hazard analysis, we obtain a small set of reusable software models for our library (see [Struss, 2013]). Of course, if we do have a more detailed model of the software, also the fault models can be more specific.

4.2 Driving Situations and Hazards

While the model captures the objective physical behavior of the system, the **functional** perspective is introduced by the selection and definition of the relevant and meaningful scenarios (driving situations) and hazard (violations of the desired functionality) for which the fault analysis is carried out. Whether a component fault causes a hazard is usually dependent of the context: if the retarder is stuck and, hence, applies a braking torque does not lead to a risk if the driver pushes the brake pedal, anyway. Therefore, hazard analysis (and FMEA) is carried out for certain different **driving situations**. From the material in our case study, we observed that there are relatively few of them. They can be characterized by the **vehicle velocity** and the **driver demand**, which is expressed by pushing the accelerator pedal or the brake pedal and selecting the gear.

The considered scenarios are normal **driving** (with or without intended acceleration), **starting**, and **braking** for both forward and backward motion. The forward ones are defined according to Table 2. Based on the documentation of the manual analysis and interviews, we introduced a distinction between "low speed" ("+") and "high speed" ("++"), which are both mapped to "+" in the physical model.

Table 2. Selected Definitions of Driving Situations

	Accelerator pedal pushed	Brake pedal pushed	Gear			Clutch pedal not pushed	v
			F	N	R		
F-start	x		x			x	0
Drive, high speed	x		x			x	++
Drive, low speed	x		x			x	+
F-brake, high speed		x	x			(x)	++
F-brake, low speed		x	x			(x)	+

Also the hazards are predefined. For the drive train, the hazards are given by deviating acceleration of the vehicle (resulting from deviating torques). Hence, the **basic hazards** are Δa=+ and Δa=-. From the perspective of FMEA, they may have a different intuitive meaning for different scenarios. For instance, Δa = + means for the (forward) drive situation that the vehicle becomes faster than intended, while for the braking scenario, the braking torque is reduced or even zero. For supporting an intuitive interpretation of the hazards, we defined them in a scenario-specific way, as illustrated by Table 3, which shows only the definitions relevant for forward situations (several values in a cell represent a disjunction).

Table 3. Selected Hazard Definitions

Hazard	Driving Situation	a	Δa
Increased acceleration	Drive, F-start	+	+
Reduced or no acceleration	Drive, F-Start	+, 0	-
Unintended deceleration	Drive	-	-
Unintended backward acceleration	F-start	-	-
Reduced or no deceleration	F-brake	-, 0	+
Increased deceleration	F-brake	-	-
Unintended acceleration	F-brake	+	+

4.4 Results of Automated Hazard Analysis

Raz'r performs the checks described in section 3 for all faults, scenarios, and hazards, and summarizes them in a table, which represents a key result of safety analysis. Table 4 shows a part of it, including software faults. With respect to the modeled component faults and the defined driving situations and hazards, the table is complete and correct. None of the entries in the table is surprising or difficult to obtain manually – but it is not the objective of this work to generate results the engineers could not produce. Instead, the goal is to automate the mechanistic part of their work. The manual production of the table costs at least tens of person hours, while the tool needs minutes. And the algorithm does neither omit scenarios or faults nor miss a hazard.

Table 4. Partial Results of Automatic Hazard Analysis for the Driving Situation "Drive"

Scenario	Part	Failure Mode	Hazard / Impact
DriveSituation	CrankShaft1	Broken	:Reduced_or_no_acceleration
DriveSituation	Clutch1	ClutchStuckOpened	:Reduced_or_no_acceleration
DriveSituation	Clutch1	ClutchStuckClosed	:>>no system level effects<<
DriveSituation	GearBox1	StuckReverse	:Unintended_deceleration
DriveSituation	GearBox1	StuckNeutral	:Reduced_or_no_acceleration
DriveSituation	GearBox1	StuckForward	:>>no system level effects<<
DriveSituation	Retarder1	RetarderStuckNotEngaged	:>>no system level effects<<
DriveSituation	Retarder1	RetarderStuckEngaged	
DriveSituation	Retarder1	RetarderStuckEngaged	:Reduced_or_no_acceleration
DriveSituation	Retarder1	RetarderStuckEngaged	:Unintended_deceleration
DriveSituation	Brakes1	StuckNotEngaged	:>>no system level effects<<
…			
DriveSituation	BrakesECU1	UntimelyCommand	:Unintended_deceleration
DriveSituation	RetarderECU1	MissingCommand	:>>no system level effects<<
DriveSituation	RetarderECU1	UntimelyCommand	
DriveSituation	RetarderECU1	UntimelyCommand	:Reduced_or_no_acceleration
DriveSituation	RetarderECU1	UntimelyCommand	:Unintended_deceleration
…			

5 Model-based Impact Analysis

The hazard analysis described above yields the consequence of faults in terms of the behavior of the CPS, in our case deviations in the motion of the vehicle, more specifically, deviation of its acceleration. Determining the **impact**

requires a model that captures the interaction of the CPS with its environment, which means in our case study, it has to represent the location and motion of the vehicle as well as other objects and to infer potential collisions. Again, this analysis is carried out for different scenarios, where scenarios in this phase need to capture different **spatial configurations** of the vehicle and other objects. Besides their (potential) spatial extension, objects have an associated type (which influences the severity of the impact). The various spatial configurations represent classes of specific real situations, such as "street with persons on sidewalk" and "approaching exit on a freeway". As a consequence, the required spatial representation has to be very abstract and qualitative, as described in the following section.

5.1 Environment Model

As opposed to extensive other work (e.g. in robot navigation) that exploits spatial reasoning for exploring trajectories of moving objects and their spatial relations and predicting collisions based on **specific** situations we need to represent archetypes of situations, possible ranges of motions, and the potential of collisions. [Dylla et al., 2007] deals with a similar task in order to represent sea navigation rules. While this domain implies degrees of freedom in 2D, our solution exploits the (trajectory on the) road as a reference, which greatly simplifies the spatial representation and reasoning.

To approach this and derive a simplified representation, we first abstract from the road as a 3D object:

- Although it may go uphill and downhill, the 3rd dimension is eliminated and only expressed as an attribute **slope** of the road, which influences the motion of the vehicle through gravitational force, which is already covered by the vehicle model.
- Although the road (or, more generally, the intended trajectory of the vehicle, as in "exiting from a freeway") may have **curves**, which influence the impact (e.g. at high speeds), we also turn this into a (Boolean) attribute of the road, indicating whether the **curvature** is significant or not
- Then we apply a **transformation** $\Re^2 \rightarrow \Re^2$ by turning the vehicle trajectory into one coordinate axis, σ, and the orthogonal distance from the road into the other coordinate, δ, with the initial location of the vehicle in the origin, as illustrated by Figure 4. Unless the trajectory is a straight line, this mapping is well-defined only for a certain envelope of the trajectory which depends on its curvature. Outside this envelope, an object on the concave side of the trajectory covers an extended area in the (σ,δ)-space, which is a feature, rather than a deficiency: in reality, it is adjacent to the trajectory for a longer section and could be hit by the vehicle when it deviates from the trajectory at several positions.

Finally, we abstract this space according to the distinctions that appeared in the natural language descriptions supplied by the industrial partner, i.e. we discretize $\Re^2$ to a level that captures the qualitative distinctions needed to characterize locations and is able to infer a potential collision due to the (qualitatively) deviating motion of the vehicle. We chose the grid depicted in Figure 5. The grid is defined by qualitative positions 0 (at the vehicle), **c**lose, **m**edium, **f**ar for σ, i.e. along the vehicle trajectory, and **s**traight, **r**ight-of, **m**edium-**r**ight-of, **f**ar-**r**ight-of (and the same for left) for δ, the distance from the trajectory. The vehicle's initial position will always be in (0, s), while, for instance, pedestrians may cover the r-strip, or a median be located in the l-strip.

The environment model is specific to the analysis task. In our case study, it is a "component" that is connected to the road (retrieving its curvature attribute), the vehicle (to access its speed and acceleration (deviation)), and other objects, which have a location and type. Right now, we include obstacles (immobile objects), other vehicles (with the potential to move fast, and persons (moving slowly).

The spatial interaction is modeled by **impact range constraints** that determine the potential positions of the vehicle and the other objects after the initial situation. For instance, a slowly driving vehicle with Δa=+ may reach positions (s, c) and (s, m). A fast, braking vehicle with Δa=+ on a road with non-zero curvature covers {(s, c), (s, m), (s, f), (l, m), (l, f), (ml, f), (r, m), (r, f), (mr, f)}, i.e. it may deviated from the trajectory.

5.2 Spatial Configurations and Impacts

While the environmental model represents the physics (of motion), the focus of the analysis and the relevant aspects of safety are expressed by the scenarios and effects to be considered. The former are given by combining the driving situations and road conditions (including curvature) used in hazard analysis with spatial configurations of other objects (relative to the vehicle position), which correspond to different classes of traffic situations. For instance, "pedestrian crossing medium" is represented by

(vehicle.speed=+,
object.type="persons",
object.location={(m, s), (m, l), (m, r)},
road.curvature=0),

and "approaching freeway exit" by

(vehicle.speed=++,
object.type="obstacle",
object.location={(m, l), (f, l)},
road.curvature=1).

Finally, the impacts, i.e. effects, to be determined for this

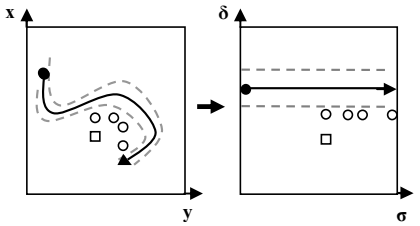


Figure 4. Transformation of the Coordinate System

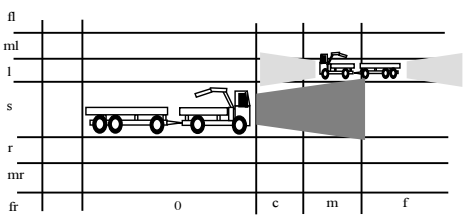


Figure 5. The Qualitative Spatial Representation

part of the analysis are modeled by **collision constraints** in the environment model: they can simply be encoded as

Equal (vehicle.position, object.position),

on the potential locations of the vehicle and of another object. If these effects are consistent with a scenario, this means the impact ranges have a non-empty intersection and, hence, a collision is possible. Impacts can be defined specific to the type of objects and, thus, determine the severity of the impact.

5.3 Results of Impact Analysis

Impact analysis is also carried out by the Raz'r FMEA engine in the same way as hazard analysis, with the impacts as relevant effects.

An example for the automatically generated results is shown in Table 5. The results have been successfully evaluated based on a set of standard situations supplied by our industrial partners. Such a gold standard for evaluation is limited and may appear subjective and vague. However, due to the nature of the task, this is the only criterion for evaluation, and we cannot expect a more rigorous and objective reference. Of course, other examples and different practice may require a revision and refinement of the current distinctions, esp. in the physical model and the spatial representation. However, so far, no evidence occurred that the current foundation for our tool would need substantial revisions.

Table 5. Partial Results of Automatic Impact Analysis for "Approaching Freeway Exit, High Speed, Braking"

Part	Failure Mode	Hazard / Impact
CrankShaft1	Broken	:collision_with_object
Clutch1	ClutchStuckOpened	:collision_with_object
Clutch1	ClutchStuckClosed	:>>no system level effects<<
GearBox1	StuckReverse	:collision_with_object
GearBox1	StuckNeutral	:collision_with_object
GearBox1	StuckForward	:>>no system level effects<<
Retarder1	RetarderStuckNotEngaged	:collision_with_object
Retarder1	RetarderStuckEngaged	:>>no system level effects<<
Brakes1	StuckNotEngaged	:collision_with_object
Brakes1	StuckEngaged	:>>no system level effects<<

6 Outlook

The results obtained have increased industrial interest in pursuing this line of research. We are currently preparing a collaborative project involving an automotive manufacturer, an automotive software engineering company, a certification company, and academic partners (representing model-based approaches from AI and software engineering) that aims at providing foundations for tools for functional safety that are compliant with the standards and processes. This will require embedding the analytic part covered here with higher-level models from design and also feeding back its results to the process of responding to severe shortcomings by developing appropriate safety functions. Steps towards formal foundations for an integration of the model-based systems and software engineering technologies will be required for this.

Acknowledgements

We would like to thank our partners from ITK for providing their domain knowledge and their patience and Alessandro Fraracci for his support. Special thanks to Oskar Dressler (OCC'M Software) for providing a very efficient implementation of the FMEA algorithm.

References

[Dobi et al., 2013] Dobi, S., Gleirscher, M., Spichkova, M., Struss, P. Model-based Hazard Analysis and Risk Assessment Technical Report TUM-I1333, Technische Universität München, 2013.

[Dylla et al., 2007] Frank Dylla, Lutz Frommberger, Jan Oliver Wallgrün, Diedrich Wolter, Berhard Nebel and Stefan Wölfl. SailAway: Formalizing Navigation Rules. In: *Proceedings of the Artificial and Ambient Intelligence Symposium on Spatial Reasoning and Communication*, AISB, p. 470-474, Newcastle upon Tyne, UK, 2007.

[Kurtoglu-Tumer-Jensen, 2010] Kurtoglu, T., Tumer, I.Y., Jensen, C.: A functional failure reasoning methodology for evaluation of conceptual system architectures. Research in Engineering Design 21.4 (2010): 209-234.

[ISO-26262, 2011] ISO, "ISO 26262", international Standard ISO/FDIS 26262, 2011. http://www.iso.org/

[Picardi et al., 2004] C. Picardi, L. Console, F. Berger, J. Breeman, T. Kanakis, J. Moelands, S. Collas, E. Arbaretier, N. De Domenico, E. Girardelli, O. Dressler, P. Struss, B. Zilbermann: AUTAS: a tool for supporting FMECA generation in aeronautic systems. In: Proceeding of the 16th European Conference on Artificial Intelligence August 22nd - 27th 2004 Valencia, Spain, pp. 750-754

[Price, 2000] Price, C. Autosteve: automated electrical design analysis. In Proceedings ECAI-2000, p.721-725, 2000

[Raz'r, 2013] http://www.occm.de/

[Struss, 2004] Struss, P.: Models of Behavior Deviations in Model-based Systems. In: Proceeding of the 16th European Conference on Artificial Intelligence August 22nd - 27th 2004 Valencia, Spain, pp. 883-887

[Struss-Fraracci-Nyga, 2011] Struss, P., Fraracci, A., Nyga, D.: An Automated Model Abstraction Operator Implemented in the Multiple Modeling Environment MOM. In: 25th International Workshop on Qualitative Reasoning, Barcelona, Spain, 2011.

[Struss-Fraracci, 2012] StrussP., Fraracci, A.: Automated Model-based FMEA of a Braking System. 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess 2012), Mexico City, 2012.

[Struss, 2013] Struss, P.: Model-based Analysis of Embedded Systems: Placing it upon its Feet instead of on its Head - An Outsider's View - In: 8th International Conference on Software Engineering and Applications (ICSOFT-EA 2013), Reykjavik, Iceland, July 29-31 2013.

Automated Generation of Diagnosis Models for ROS-based Robot Systems

Safdar Zaman and **Gerald Steinbauer**
Institute for Software Technology, Graz University of Technology, Graz, Austria
{szaman,steinbauer}@ist.tugraz.at

Abstract

Automated fault detection and repair are necessary capabilities for a robot system that claims to be fully autonomous. A model-based diagnosis system compares a model of the indented system's behavior with the observed behavior in order to detect faults at runtime. A diagnosis model is a set of rules describing the correct behavior of the robot system while observations are the output of a monitoring system comprised by different observers. In this paper we present an approach for an automated generation of diagnosis models and observers. During a learning phase the approach observes the running target system and identifies the important processing nodes and their communication channels and behaviors. Moreover, the approach looks for particular properties of these nodes and statistical relations among the processed data that can be utilized in a diagnosis model. Finally, the approach generates a set of observers that can be used in combination with the diagnosis model to form a diagnosis system for ROS-based robot systems. Experiments using a robot carrying out mapping tasks show that the proposed approach is able to generate with minimal user interaction diagnosis models and observers that are able to detect faults of the robot system at runtime.

1 Introduction

For achieving a given task autonomously a robot system use a number of different hardware (e.g. sensors or actuators) and software (e.g. navigation and planning modules) components. During accomplishing its task it is quite likely that a component shows an undesired behavior that can be caused by a wide range of faults such as defective hardware or software deadlocks. This phenomena is caused by the complex interactions within the robot system and the non-deterministic interaction with the dynamic environment. In order to be able to automatically cope with such problems it is necessary to have a monitoring system that is not only able to detect such faults but is also able to repair them at runtime. If a model-based diagnosis approach is used one needs a model of the correct system behavior. This behavior model is then compared to the behavior observed at runtime in order to detect and localize faults.

Such a diagnosis model can be acquired using three basic approaches. The first approach is to reuse requirements or engineering models that are already available if for instance a model-driven development process is used [1]. If no reusable models are available diagnosis models can be created by hand. While this second approach is quite widespread it is cumbersome and error-prune in particular for complex systems. In this paper we follow a third approach that learns diagnosis models on-line during a controlled learning phase.

In order to acquire a model on-line one has to have some mechanisms to analyze the computation and interaction of the system's components. The analyzed data describes the behavior of the robot system at runtime and can be used to generate a diagnosis model. A basic assumption needed for this approach to work is that the system produces no faults during the learning phase. Moreover, it has to be assumed that the robot show all possible behaviors during the learning phase in order to generate a complete diagnosis model.

The model generation approach presented in this paper is based on the diagnosis and repair system for ROS-based robot systems presented in [2]. The Robot Operating System (ROS) [3] is a popular open-source software framework for robot application. It follows the concept of a computation graph realized by computation nodes (i.e. processes) that communicate using a publisher-subscriber mechanism. The diagnosis and repair system utilizes this easy observable communication mechanism to detect and localize faulty nodes. In previous work diagnosis models describing the communication behavior and other properties of nodes were handcrafted. Major parts of the models are information about which nodes have to be active, their properties (e.g. average cpu usage), how these nodes communicate (e.g. average communication frequency) and input-output relations of nodes. Moreover, a set of observer is generated that observes the parameter of the model at run-time.

In this paper we propose to use information recorded from a system actually performing a complete task to extract the diagnosis model automatically. ROS supports this approach because it allows easy access to information about the computation graph, the involved nodes, the communication between nodes and the content (data) of exchanged messages. The recorded information belong to three groups: (1) running nodes and their properties, (2) communication patterns and (3) exchanged data. The first group is directly available form ROS and can directly transfered to rules in the model (e.g. which nodes have to run). For the second group ROS only provides information about exchanged messages. Here

a statistical analysis has to be done in order to detect communication patterns (e.g. which nodes communicate regularly). Finally, the values contained in the messages can be correlated to detect functional dependencies (e.g. if one value increases another one has to increase as well). Here we follow a qualitative reasoning approach [4].

The remainder of the paper is organized as follows: Related research on fault detection and repair in the robotics domain is reviewed in Section 2. The next section briefly introduce the diagnosis and repair system this work is based on. In Section 4 the approach for automated model generation is explained in detail. In the following section experimental results using a mapping scenario are presented. Section 6 summarizes the presented approach and discusses future work.

2 Related Research

A lot of research on diagnosis and repair for robots has been conducted during the last decades. The approaches basically differ in the application domain (software, hardware, behavior, or an integration of all), in the methodology used (FDI or model-based) and if diagnosis and repair are combined. In [5] the authors presented a model-based approach for diagnosing faults of robot control software which use a model of the communication between components. The approach was based on a CORBA-based communication framework. For the work presented in this paper the ROS framework [3] had been used. Automated learning of the communication models for robot navigation software is presented in [6]. The approach identifies different types of communication patterns which are used in the diagnosis model. We reuse this idea. But using an intelligent automated selection we avoid the manual setting of many parameters which is necessary in that approach. In the context of hardware diagnosis and repair the authors of [7] presented a mechanisms for fault diagnosis and repair of robot drives at runtime. It provides a control framework which is capable of reconfiguring the control functions of the drive based on detected faults. In [2] an integrated diagnosis and repair approach for ROS-based robot system is presented. The approach uses a model-based approach and is able to deal with faults in software and hardware. Within this paper we present an extension to this approach.

The authors of [8] presented an hybrid diagnosis approach for a printer system where mode estimation for continuous sensor measurements was combined with a decision tree using discrete components modes. The diagnosis approach presented in this paper is related to this work because it use correlations of signals to generate input for a discrete model-based approach. The authors of [9] and [10] presented statistical learning techniques for models of the communication within robot systems. The presented techniques allows for estimating a probability distribution of the internal data exchange and communication in a robot system. This approach is similar to the presented because it use statistical information about the occurrence of messages. We extend this approach by integrating also the content of the messages. In [11] an approach for on-line diagnosis of components of autonomous system is presented. The approach compares pairs of sensor signals in order to detect faulty components. This approach is similar to the presented approach but it need to know the structural model in advance. Moreover, the signals are only linear correlated. In contrast our approach learns the structural model automatically and correlates the signals qualitatively over a period of time. A different approach using particle filters for estimation the mode of robot hardware was presented in [12]. In the domain of diagnosis in multi robot systems the work in [13] uses communicating automatons where the work in [14] uses causal models and a hypotheses generation and testing strategy. The latter moreover provides pre-defined repair action and the learning of new fault types.

3 Prerequisites

The presented work is an extension to model-based diagnosis and repair approach for robot systems presented in [2]. The system depends on the node and communication concept of ROS. ROS identify communication channels by so called topics. An overview of the system is shown in Figure 1.

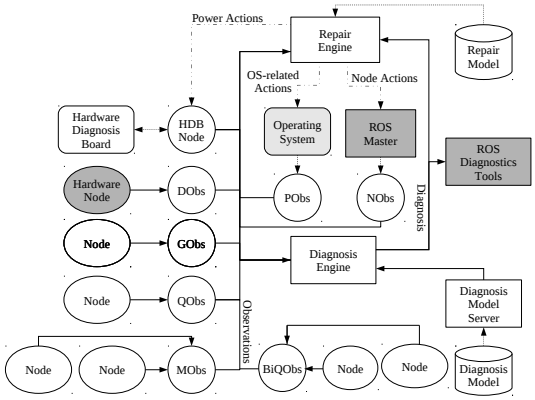


Figure 1: Overview of the model-based diagnosis and repair system.

It comprises the following components: a set of observers, a diagnosis model server, a diagnosis engine, and a repair engine. Observers are entities that monitor a particular behavior or property of the system. Each observer is of a particular type. Possible types are:

- **general observers (GObs)** observe a particular topic t for regular communication at a given frequency f_t
- **node observers (NObs)** monitor if a particular node n is running
- **diagnostic observers (DObs)** integrate existing ROS diagnostics information, hardware status information is mapped to observations
- **hardware observers (HObs)** process information of a hardware diagnostic board (e.g. current measurements)
- **qualitative observers (QObs)** monitor qualitative relations of values in messages, related value follow a common trend
- **multiple observers (MObs)** monitor conditional communication of inputs and outputs of a node n, i.e. an output is triggered by an input
- **property observers (PObs)** monitor if a particular property of a node n exceeds its limits (e.g. memory)

Each observer is implemented as ROS node and publish its observations on topic */observations*. Observations

are first order logic (FOL) sentences like $ok\_topic(t_i)$ or $\neg running(n_i)$ describing that communication via topic t_i is working correct or node n_i does not run. Please note that observer only give an alarm if the violation persist for a significant time in order to deal with tiny fluctuations in the system (for details see [15]).

The diagnosis model server provides the diagnosis model system-wide. This mechanism allows to change the diagnosis model at runtime. The diagnosis engine follows the diagnosis principles presented in [16] and derives diagnoses using the provided diagnosis model and observations coming from the observers. We use the representation presented in [17]. The engine publishes diagnoses on topic */diagnosis* in the form of a set bad comprising the faulty components and a set $good$ with the working components. If a diagnosis with a non-empty set bad is derived the repair engine gets invoked.

It is a action planner that uses a description of the possible repair actions in the Planning Domain Definition Language (PDDL) [18] to find sequence of repair actions that brings the system back in the nominal state. Possible repair action are restart of a node, a power-up cycle for a hardware component or a reconfiguration of a node. During the execution of the repair plan the diagnosis engine is suspended to avoid transient diagnoses. The ROS-based diagnosis and repair system is publicly available as open-source[1].

In the remainder of the paper we focus on the automated generation of the diagnosis model. An automated generation of the repair actions is beyond the focus of this paper.

4 Diagnosis Model Generation

In this section we present in detail the approach for the automated generation of diagnosis model and the instantiation of related observers.

4.1 Recording the Running System

The input to the generation and instantiation step is recorded during a fault-free execution of tasks by the robot system and comprises three parts: (1) the computation graph, (2) the communication via all topics and (3) further property observation.

Definition 1 A topic-node relation is a tuple $r_i = \langle t_i, N_{t_i} \rangle$ where t_i is a topic and N_{t_i} is a set of nodes publishing on or subscribing to topic t_i.

The computation graph can be directly obtained using ROS system functions.

Definition 2 A computation graph is a tuple $G = \langle N, T, P, S \rangle$ where $N = \langle n_1, ..., n_{k_N} \rangle$ is the set of running nodes, $T = \langle t_1, ..., t_{k_T} \rangle$ is the set of topics, $P = \langle p_1, .., p_{k_P} \rangle$ is the set of publishing topic-node relation and $S = \langle s_1, .., s_{k_S} \rangle$ is the set of subscribing topic-node relation. For better readability we assume access functions for tuples in the form $e(t)$ for accessing the entry e of tuple t.

A communication via a topic t_i is a time-ordered list of exchanged messages that can be simply obtained by subscribing to topic t_i.

[1] http://www.ros.org/wiki/tug_ist_model_based_diagnosis

Definition 3 The communication (CO) for a topic t_i is a time-ordered list $M_{t_i} = \{m^1_{t_i}, ..., m^{k_{M_{t_i}}}_{t_i}\}$ where $m^j_{t_i}$ is a tuple $\langle \nu, t \rangle$ with $\nu = \{v_1, ..., v_{k_{\nu_{t_i}}}\}$ a set of atomic values in the exchanged message and t is the time of the occurrence of the message. Atomic values are data types that can not decomposed anymore such as integer or floats. The set of all communications is denoted as M. We assume that message layouts do not change during runtime. Therefore, we treat the number of values $k_{\nu_{t_i}}$ in a message for topic t_i as a constant and define a function V_c that extract from a communication M_{t_i} the list of the $j^{th} value, j \in \{1, ..., k_{\nu_{t_i}}\}$, and their occurrence time: $V_c : CO \times \mathbb{N}^+ \rightarrow \{\langle \mathbb{R}, \mathbb{R} \rangle\}$. Finally, we define two functions Γ and Δ that extract from a communication M_{t_i} the list of the occurrences respectively the time difference to the previous occurrence: $\Gamma, \Delta : CO \rightarrow \{\mathbb{R}\}$. The time difference for the first occurrence is defined as 0.

Property observations for a node are time-ordered list of real numbers. Currently two kinds of observations types are supported: (1) cpu usage and (2) memory usage. As nodes are processes these observations can be easily acquired using OS functionality (e.g. proc file system).

Definition 4 A property observation (PO) for a node n_i is a time-ordered list $\Pi_{n_i} = \{\pi^1_{n_i}, ..., \pi^{k_{\Pi_{n_i}}}_{n_i}\}$ where $\pi^j_{n_i}$ is a tuple $\langle \pi, \tau, t \rangle$ with $\pi \in \mathbb{R}$ the quantity of the observed property of type $\tau \in \{\tau_{CPU}, \tau_{MEM}\}$ at time t. The set of all property observations is denoted as Π. Moreover, we define a function V_p that returns for a property observation a set of all values of a particular type: $V_p : PO \times \tau \rightarrow \{\mathbb{R}\}$.

4.2 Instantiating the Observers

The recording of the running system provides a computation graph G, a set of communications M and set of property observations Π. This information are fed into Algorithm 1 to derive the sets of observers that have to be instantiated to observe the running system during the diagnosis process.

Algorithm 1: $instantiateObs(G, M, \Pi)$

input : G ... the computation graph
input : M ... the communication set
input : Π ... the property observation set
output: a set of node observers O_n
output: a set of general observers O_g
output: a set of property observers O_p
output: a set of qualitative observers O_q

$O_n = \emptyset$,= $O_g\emptyset$,$O_q = \emptyset$
foreach $n_i \in N$ **do**
  $O_n = O_n \cup NObs(n_i)$
end
foreach $M_{t_i} \in M$ **do**
  $\bar{\Delta} = mean(\Delta(M_{t_i})), \sigma = stddev(\Delta(M_{t_i}))$
  if $\bar{\Delta}/\sigma > \alpha$ **then**
    $O_g = O_g \cup GObs(t_i, \bar{\Delta}, \sigma)$
  end
end
foreach $\Pi_{n_i} \in \Pi$, $t \in \tau$ **do**
  $\bar{\Pi} = mean(V_p(\Pi_{n_i}, t)), \sigma_\Pi = stddev(V_p(\Pi_{n_i}, t))$
  $O_p = O_p \cup PObs(n_i, t, \bar{\Pi}, \sigma_\Pi)$
end
$O_q = \cup instantiateQObs(M)$

First the algorithm instantiates for each node a node observer and stores them in the set O_n (lines 2-4). Then the algorithm calculates for each communication via a topic the mean and standard deviation of the time difference of successive messages. Using the heuristic that the fraction of the mean and the standard deviation is above a certain threshold α for topics communicating on a regular basis general observer are instantiated for such topics and stored in O_g (lines 5-10). For each node where a property observation is available a property observer is instantiated with mean and standard deviation of the related property observation (lines 11-14). Qualitative observer are instantiated using Algorithm 2 (line 15).

Algorithm 2: $instantiateQObs(M)$

input : M ... the communication set
output: a set of qualitative observers O_q

1 $O_q = \emptyset$
2 **foreach** $\{\langle t_i, t_j\rangle | M_{t_i} \in M \wedge M_{t_j} \in M\}$ **do**
3 $\quad \bar{\Delta}_i = median(\Delta(M_{t_i}))$,$\bar{\Delta}_j = median(\Delta(M_{t_j}))$
4 $\quad$ **foreach** $l \in \{1, ..., k_{\nu_{t_i}}\}$,$m \in \{1, ..., k_{\nu_{t_j}}\}$,$i \neq j \vee l \neq m$ **do**
5 $\quad\quad$ **if** $correlated(M_{t_i}, l, \bar{\Delta}_i, M_{t_j}, m, \bar{\Delta}_j)$ **then**
6 $\quad\quad\quad O_q = O_q \cup QObs(t_i, t_j, l, m, \bar{\Delta}_i, \bar{\Delta}_j, 0, 0)$
7 $\quad\quad$ **end**
8 $\quad\quad$ **if** $correlated(I(M_{t_i}), l, \bar{\Delta}_i, M_{t_j}, m, \bar{\Delta}_j, 0, 1)$ **then**
9 $\quad\quad\quad O_q = O_q \cup QObs(t_i, t_j, l, m, \bar{\Delta}_i, \bar{\Delta}_j)$
10 $\quad\quad$ **end**
11 $\quad\quad$ **if** $correlated(M_{t_i}, l, \bar{\Delta}_i, I(M_{t_j}), m, \bar{\Delta}_j)$ **then**
12 $\quad\quad\quad O_q = O_q \cup QObs(t_i, t_j, l, m, \bar{\Delta}_i, \bar{\Delta}_j, 1, 0)$
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 **end**

The algorithm first calculates the average communication differences for all pairs of topics (lines 2). Then it is checked if both original or one original and one integrated communication is correlated (lines 4-14). If two communications are qualitatively correlated a qualitative observer is instantiated where the last two parameter determines if a communication will be integrated during observation. $I(M)$ denotes the communication where the values in M are replaced by the sum of itself and all successive values. If two values in communications are qualitatively correlated is determined using Algorithm 3.

The algorithm determine the qualitative correlation of two value by comparing their qualitative trend. Here we follow the ideas presented in [15]. In technical systems and in particular in robot system related values can hardly be matched on an absolute scale. For instance the orientation measured by a compass and the odometry may start at different absolute angles. But the qualitative trend (e.g. increase, decrease, constant) of both have to be the same for related values. Moreover, values of a physical system are noisy. Therefore, we use a sliding window and linear regression for calculating the slopes of a list of values. The sliding windows ws_i and ws_j (line 1) ensures that in average C occurrences are used for the linear regression for the l^{th} value of topic t_i (lines 3-6) respectively for the m^{th} value of topic t_j (lines 7-10).

According to [15] a threshold b is necessary to classify a slope (trend) as increasing, decreasing or constant denoted by the symbols $+$,$-$ and 0. In order to be able to extract this parameter automatically we introduce two distinct parameter b_+ and b_- for increasing and decreasing trends (lines 11-12). Assuming that the recorded data sufficiently represents the true probability distribution of trends we set the threshold for b_+ and b_- to the median of all positive slopes respectively all negative slopes. These parameters are finally used to classify the trends including checking higher-order derivatives as proposed in [15] (lines 13-14). Finally, the matches of the qualitative trends of both values are calculated with c representing the number of checked trends and m representing the number of matches (lines 15-31).

Algorithm 3: $correlated(M_{t_i}, l, \bar{\Delta}_i, M_{t_j}, m, \bar{\Delta}_j)$

input : M_{t_i} ... the communication of topic t_i
input : l... use l^{th} value of topic t_i
input : $\bar{\Delta}_i$ ... median message time difference of topic t_i
input : M_{t_j} ... the communication set of topic t_j
input : m ... use m^{th} value of topic t_j
input : $\bar{\Delta}_j$ ... median message time differences of topic t_j
output: $true$ if t_i and t_j are qualitatively correlated, $false$ otherwise

1 $ws_i = C\bar{\Delta}_i$, $ws_j = C\bar{\Delta}_j$
2 $s_i^l = \emptyset$, $s_j^k = \emptyset$
3 **foreach** $v \in V_c(M_{t_i}, l)$ **do**
4 $\quad s = linreg(\{v' \in V_c(M_{t_i}, l) | time(v') \in [time(v) - ws_i/2, time(v) + ws_i/2]\})$
5 $\quad s_i^l = s_i^l \cup \langle time(v), s\rangle$
6 **end**
7 **foreach** $v \in V_c(M_{t_j}, m)$ **do**
8 $\quad s = linreg(\{v' \in V_c(M_{t_j}, m) | time(v') \in [time(v) - ws_i/2, time(v) + ws_i/2]\})$
9 $\quad s_i^m = s_i^m \cup \langle time(v), s\rangle$
10 **end**
11 $b_i^+ = median(\{val(s) | s \in s_i^l \wedge val(s) > 0\})$, $b_j^+ = median(\{val(s) | s \in s_j^m \wedge val(s) > 0\})$,
12 $b_i^- = median(\{val(s) | s \in s_i^l \wedge val(s) < 0\})$, $b_j^- = median(\{val(s) | s \in s_j^m \wedge val(s) < 0\})$
13 $q_i = trend(s_i, b_i^+, b_i^-, ws_i)$
14 $q_j = trend(s_j, b_j^+, b_j^-, ws_j)$
15 $c = 0, m = 0$
16 **foreach** $q \in q_i$ **do**
17 $\quad q_j^- = min_{q' \in q_j}(time(q) - time(q'))$
18 $\quad q_j^+ = min_{q' \in q_j}(time(q') - time(q))$
19 $\quad$ **if** $match(q, q_j^-, q_j^+)$ **then**
20 $\quad\quad$ m = m + 1
21 $\quad$ **end**
22 $\quad$ c = m + 1
23 **end**
24 **foreach** $q \in q_j$ **do**
25 $\quad q_i^- = min_{q' \in q_i}(time(q) - time(q'))$
26 $\quad q_i^+ = min_{q' \in q_i}(time(q') - time(q))$
27 $\quad$ **if** $match(q, q_i^-, q_i^+)$ **then**
28 $\quad\quad$ m = m + 1
29 $\quad$ **end**
30 $\quad$ c = m + 1
31 **end**
32 **return** $(m/c) > Q$

Because in general the number of occurrences of trends and their time is not equal for two topics we use an interpolation for the qualitative trend. If the occurrence of a qualitative trend q of topic t_i has to be matched we derive the closest occurrence of a trend in topic t_j before and after q.

We interpolate the symbol q has to be matched again using Table 1. It is reasonable to assume that for instance if the corner symbols are $+$(increasing) and $-$(decreasing) there is a middle area in the interpolation with symbol 0 (constant). Finally, a match is reported if the ratio between the number of matches and the number of total checks is greater than a parameter Q (line 32).

4.3 Generating the Diagnosis Model

As already mentioned above we follow the diagnosis principles form [16]. Moreover, we use a more efficient model representation based on Horn clauses [17]. We extract the diagnosis model from the information collected trough the recording phase and the instantiated observer using Algorithm 4. The algorithm gets the computation graph (G) and the sets of observers (O_n,O_g,O_p,O_q) and returns a set of Horn clauses M forming the diagnosis model.

Algorithm 4: $generateModel(G, O)$

input : G ... the computation graph
input : set of instantiated node observers O_n
input : set of instantiated general observers O_g
input : set of instantiated property observers O_p
input : set of instantiated qualitative observers O_q
output: a set of clauses M

1 $M = \emptyset$
2 **foreach** $o \in O_n$ **do**
3 $\quad M = M \cup \{\neg AB(node(o)) \rightarrow running(node(o))\ \}$
4 **end**
5 **foreach** $n \in N$ **do**
6 $\quad P' = \{t' \in T | (\exists p \in P.node(p) = n \wedge t' \in topic(p)) \wedge$
7 $\quad \exists o \in O_g \wedge topic(o) = t'\}$
8 $\quad$ **foreach** $t'' \in P'$ **do**
9 $\quad\quad S' = \{t''' \in T | (\exists s \in S.node(s) = n \wedge$
10 $\quad\quad t''' \in topic(s)) \wedge \exists o \in O_g \wedge topic(o) = t''\}$
11 $\quad\quad M = M \cup \{\neg AB(n) \wedge \bigwedge_{t' \in S'} ok\_topic(t'') \rightarrow$ $ok\_topic(t')\}$
12 $\quad$ **end**
13 **end**
14 **foreach** $o \in O_p$ **do**
15 $\quad M = M \cup \{\neg AB(node(o)) \rightarrow$ $ok\_prop(node(o), type(o))\}$
16 **end**
17 **foreach** $o \in O_q$ **do**
18 $\quad N_q = \{n \in N | \exists p \in P.node(p) = n \wedge$
19 $\quad t_1(o) \in topic(p) \vee t_2(o) \in topic(p)\}$
20 $\quad T' = \emptyset$
21 $\quad$ **if** $\exists o \in O_g.t_1 = topic(o)$ **then**
22 $\quad\quad T' = T' \cup t_1$
23 $\quad$ **end**
24 $\quad$ **if** $\exists o \in O_g.t_2 = topic(o)$ **then**
25 $\quad\quad T' = T' \cup t_2$
26 $\quad$ **end**
27 $\quad M = M \cup \{\bigwedge_{n \in N_q} \neg AB(n) \wedge \bigwedge_{t \in T'} ok\_topic(t) \rightarrow$ $ok\_match(t_1(o), v_1(o), t_2(o), v_2(o))\}$
28 **end**

The extraction of the clauses can be done straight forward using the information and relations contained in graph and the observers. The algorithm starts with an empty set of clauses (line 1). For each node with a node observer we add a clause that states that if a node is working correctly there should be a process for it (lines 2-4). We use the common nomenclature that the literal $\neg AB(c)$ denoted that component c is working correctly.

Moreover, we add for each node n with a topic t' the node is publishing on and a general observer (check for regular communication) for t' a clause that specifies that this observer have to report $ok\_topic(t')$ if the node n works and all its subscribed topics t'' that have an observer report $ok\_topic(t'')$ as well (line 5-13). These clauses cover the input/output relations of nodes. For each property observer we add a clause that if a node n works all its observed properties of particular types have to be within the specified boundaries (line 14-15). The literal $ok\_prop(n, t)$ denoted that the property of type t is ok for node n.

Finally, we specify that if two topics t_1 and t_2 are qualitative correlated and there is a related observer the two topics have to qualitatively match (line 27-23). Where the literal $ok\_match(t_1, l, t_2, m)$ that the l^{th} value of topic t_1 is qualitatively matching with the m^{th} value of topic t_2. For instance in a fault-free system the yaw angle reported by the odometry should have the same trend than a yaw reported by an IMU.

5 Evaluation

In order to evaluate the proposed diagnosis model learning we conducted a series of experiments. We used a mobile robot in a teleoperated mapping scenario. Using the proposed learning approach and a fault-free task execution we obtained the diagnosis model. This automated generated diagnosis model was used for diagnosis in another fault-free and a faulty task execution in order to evaluate if the system reports any false positives or negatives. Finally, we evaluated the sensitivity of the approach on changes of parameters.

5.1 Experimental Setup

For the evaluation we used a Pioneer DX3 robot equipped with a Sick LMS 200 laser scanner and a XSense MTi IMU. The robot was controlled by a standard ROS installation. For mapping the open-source SLAM implementation gmapping[2] was used. The robot was teleoperated by joystick while mapping corridors in a building of a size of 20 m x 14 m. For the fault-free task execution we run on solid flat floor. For the faulty task execution we ran sporadically over loose paper. When the robot drove over this the wheels non-deterministically lose contact and the odometry turns faulty.

5.2 Model Validation

For validation of the model generated we conducted a fault-free execution of the mapping task. This led to recorded data of 244.80 seconds. Using the proposed approach this data were transferred into a set of 39 observers (7 node observers, 17 general observers, 14 property observers and 1 qualitative observer). The computation graph is depicted in Figure 2.

For the instantiation of the qualitative observer 21 values and their integration were correlated. Values were for instance roll, pitch and yaw from odometry or IMU. The correlation were calculated using a Matlab script. The critical parameter Q was set to 0.8. Figure 3 shows the correlation of two lists of values. Based on the computation graph and the observer a diagnosis model comprising 44 clauses was

[2] see http://www.openslam.org/gmapping

$s(q^-)$	$s(q^+)$	$\bar{s}(q)$			
		$t(q) \in [t(q^-), t(q^-) + \Delta_t/3)$	$t(q) \in [t(q^-) + \Delta_t/3, t(q^-) + \Delta_t/2)$	$t(q) \in [t(q^-) + \Delta_t/2, t(q^-) + 2\Delta_t/3)$	$t(q) \in [t(q^-) + 2\Delta_t/3, t(q^+)]$
$-$	$-$	$-$	$-$	$-$	$-$
$-$	0	$-$	$-$	0	0
$-$	$+$	$-$	0	0	$+$
0	$-$	0	0	$-$	$-$
0	0	0	0	0	0
0	$+$	0	0	$+$	$+$
$+$	$-$	$+$	0	0	$-$
$+$	0	$+$	$+$	0	0
$+$	$+$	$+$	$+$	$+$	$+$

Table 1: Interpolation for qualitative trend for time $t(q)$ between the times $t(q^-)$ and $t(q^+)$ where $\Delta_t = t(q^+) - t(q^-)$).

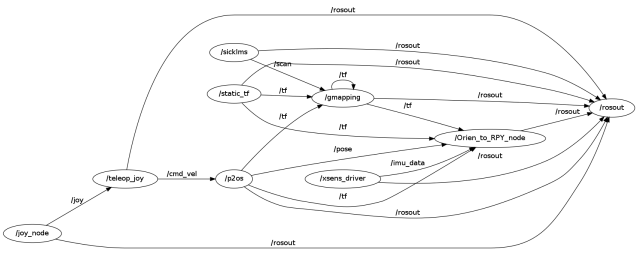


Figure 2: The computation graph of the fault-free task execution.

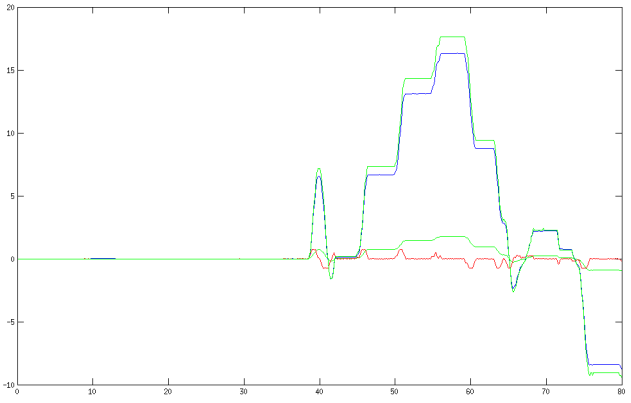


Figure 3: The correlation of the integration (green) of the angular velocity around the z-axis of the IMU (red) an d the odometry yaw (blue).

generated. The instantiation of the observer and the generation of the model took 1782 seconds on a Lenovo X201 notebook (2.4 Ghz cpu, 4 Gb memory). We denote this reference model as M_A.

During another fault-free task execution of 264 seconds we run the diagnosis and repair system of [2] using the generated model. The system reported 0 diagnosis during the execution. This was the expected value as no faults were present. No false positives were expected for a valid model. A summary of the false positives is shown in Table 2. Using the same model we conducted another task execution where the robot drove 3 times over papers. The diagnosis system reported 2 faults at the correct time exactly on the papers. This is as well a quite good value for a valid model. The reported values justify that model M_A is a valid diagnosis model for the robot system executing the mapping task.

model	Q	task execution	# faults reported	# true faults	false positives	false negatives
M_A	0.8	fault-free	0	0	0	0
M_A	0.8	on paper	2	3	0	1
M_B	0.9	fault-free	0	0	0	0
M_B	0.9	on paper	0	3	0	3
M_C	0.6	fault-free	1	0	1	0
M_C	0.6	on paper	2	3	0	1

Table 2: Number of false positives and negatives reported using different models.

5.3 Sensitivity on Parameters

One advantage of the proposed approach is that it minimizes user interaction like the specification of structures or parameters such as the structural model in [11] or the slopes in [15]. But the critical parameter Q which specifies the percentage of qualitative matches for two values necessary to be declare as correlated has to be still chosen manually. In order to evaluate the influence of the parameter Q on the quality of the model we generated two further models M_B and M_C using $Q = 0.9$ and $Q = 0.6$ respectively. Models M_B and M_C comprised 0 respectively 6 qualitative observers. We conducted one fault-free and one faulty task execution each together with two models. The collected false positives and negatives for these two models are shown in Table 2. The results show that model M_B is less sensitive to faults. It showed 3 false negatives. This fact came from the higher value for Q which led to fewer qualitative observers. In fact the important qualitative observer for the related yaw values of odometry and IMU vanished because of the high requested correlation factor. This observer is crucial to be able to detect faulty odometry. In contrast model M_C reported 1 false positives. This fact showed that the model is more sensitive to faults. This came from the fact that because of the lower required correlation factor additional relations that did not exist in reality were added. For instance a qualitative observer for the values of the x position of the robot and the rotation of the IMU was wrongly added.

Moreover, we varied the value α in order to investigate its influence on the model. We took the optimal value for Q 0.8 and set α to 10.0, 2.5 and 0.5. This led to 2 respectively 15 and 20 general observers. Then we injected a fault (increased message rate) into the p2os node. The models generated with $\alpha = 2.5$ respectively $\alpha = 0.5$ were able to detect this fault. While the model generated with $\alpha = 10.0$ was not able to detect this fault because the necessary general observer for the regular topic pose was not integrated into the model.

Finally, we introduced further faults into the system (kill software nodes, artificially increased cpu usage). All gen-

erated models were able to detect these faults because node and property observer are always generated for all nodes.

6 Conclusion and Future Work

In this paper we presented an approach for automated generation of observers and diagnosis models for robot systems from recorded task executions. The approach is an extension to the ROS-based diagnosis and repair framework presented in [2]. The approach extracts with a minimum of user interaction a structural model of the robot system. Moreover, using statistical learning correlations in the computation are extracted which can be used in the diagnosis model. Preliminary experiments with different training and test runs show that the approach is able to generate a valid model.

Although, the approach needs only a minimum set of user-specified parameters there is still need for future work in order to automate the estimation of crucial parameters such as Q. Moreover, it has to be investigated how the approach scale with the size of the robot system. In particular, a more efficient implementation and more intelligent treatment of the value correlations have to be done. Finally, the automated modeling stops currently with the diagnosis model. It will be very interesting to learn the repair model as well. Possibly, a combination of fault diagnosis and analyzing the user's reaction to faults can lead to such models. Finally, currently we demand that learning runs are fault-free. This is a somehow strong requirement. Therefore, we will investigate how much influence faults in the learning phase have on the quality of the learned model.

7 Acknowledgment

Safdar Zaman gratefully acknowledges the support from Higher Education Commission (HEC) of the government of Pakistan funding his PhD studies at Graz University of Technology in Austria.

References

[1] J. F. Broenink, M. A. Groothuis, P. M. Visser, and M. M. Bezemer. Model-driven robot-software design using template-based target descriptions. In *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications: How to modify and enhance commercial controllers, Anchorage, Allaska, USA*, pages 73–77, May 2010.

[2] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran. An Integrated Model-Based Diagnosis and Repair Architecture for ROS-Based Robot Systems. In *IEEE International Conference on Robotics and Automation (ICRA-2013)*, Karlsruhe, Germany, 2013.

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source perating system. In *ICRA Workshop on Open Source Software in Robotics*, 2009.

[4] Bert Bredeweg and Peter Struss. Current Topics in Qualitative Reasoning. *AI Magazine*, 24(4):13–16, 2004.

[5] Gerald Steinbauer, Martin Mörth, and Franz Wotawa. Real-time diagnosis and repair of faults of robot control software. In *International RoboCup Symposium*, volume 4020 of *Lecture Notes in Computer Science*, Osaka, Japan, 2006. Springer.

[6] Alexander Kleiner, Gerald Steinbauer, and Franz Wotawa. Towards Automated Online Diagnosis of Robot Navigation Software. In *First International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2008)*, volume 5325 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2008.

[7] Mathias Brandstötter, Michael Hofbaur, Gerald Steinbauer, and Franz Wotawa. Model-based fault diagnosis and reconfiguration of robot drives. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, 2007.

[8] F. Zhao, X. Koutsoukos, H.Haussecker, J. Reich, and P. Cheung. Distributed monitoring of hybrid systems: A model-directed approach. *International Joint Conf on Artificial Intelligence (IJCAI01)*, 2001.

[9] R. Golombek, S. Wrede, M. Hanheide, and M. Heckmann. Learning a probabilistic self-awareness model for robotic systems. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[10] R. Golombek, S. Wrede, M. Hanheide, and M. Heckmann. A method for learning a fault detection model from component communication data in robotic systems. In *Seventh IARP Workshop on Technical Challenges for Dependable Robots in Human Environments*, Toulouse, France, 2010.

[11] Eliahu Khalastchi, Meir Kalech, and Lior Rokach. Sensor fault detection and diagnosis for autonomous systems. In *The 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS2013)*, 2013.

[12] Vandi Verma, I. Verma, Geoff Gordon, Reid Simmons, and Sebastian Thrun. Particle Filters for Rover Fault Diagnosis. In *IEEE Robotics & Automation Magazine special issue on Human Centered Robotics and Dependability*, 2004.

[13] Roberto Micalizio, Pietro Torasso, and Gianluca Torta. On-line monitoring and diagnosis of a team of service robots: A model-based approach. *AI Communications*, 19(4):313–340, December 2006.

[14] Lynne E. Parker and Balajee Kannan. Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *IEEE International Conference on Intelligent Robots and Systems (IROS*, 2006.

[15] Alexander Kleiner, Gerald Steinbauer, and Franz Wotawa. Using qualitative and model-based reasoning for sensor validation of autonomous robots. In *Twentieth International Workshop on Principles of Diagnosis (DX 2009)*, Stockholm, Sweden, 2009.

[16] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 95, 1987.

[17] Bernhard Peischl and Franz Wotawa. Model-based Diagnosis or Reasoning From First Principles. *IEEE Intelligent Systems*, 18(3):32–37, 2003.

[18] Craig Knoblock, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David E Smith, Ying Sun, and Daniel Weld. Pddl the planning domain definition language. *AIPS-98 Competition Committee*, 78(4):1–27, 1998.

A Hybrid Approach for Fault Detection and Diagnosis in Autonamous Systems

Eliahu Khlastchi and **Meir Kalech** and **Lior Rokach**
Ben-Gurion University of the Negev, Beer-Sheva, Israel
email: {khalastc, kalech, liorrk}@bgu.ac.il

Abstract

One of the challenges of fault detection in the domain of autonomous systems is the handling of unlabeled data, meaning, most data sets are not recognized as normal or faulty. This fact makes it very challenging to be used as a training set such that learning algorithms would produce a successful fault detection model. Traditionally unsupervised algorithms try to address this challenge. In this paper we present a hybrid approach that combines unsupervised and supervised methods. An unsupervised approach is utilized for labeling a training set, and then by a standard supervised algorithm we build a fault detection and diagnosis model that is much more accurate. We show promising results on simulated and real world domains.

1. Introduction

Autonomous systems such as Unmanned Vehicles (UVs) or robots are susceptible to a variety of hardware and software faults. These faults might lead to mission failure or even endanger the safety of the expensive system or its environment. For example, a pitot-static system failure in an Unmanned Aerial Vehicle (UAV) might lead to a crash.

To continue operate autonomously, the system must have an accurate fault detection mechanism. Upon fault detection a diagnosis process can be triggered and a decision on how to continue can be made. An accurate fault detection mechanism presents several challenges in the domain of autonomous systems: (1) there is a great variety of faults, such as stuck value, drift or abrapt, (2) a fault expression can span over time, (3) a fault should be detected as quickly as possible, online, and with high accuracy, (4) a fault detection mechanism should be kept light, finally (5) this domain is also characterized for having unlabeled data.

The Diagnosis should be accurate as well. A diagnosis report should return a minimal set of components that includes the root cause of the fault.

Three general approaches are usually used for fault detection and diagnosis: Knowledge-Based systems, Model-Based, and Data Driven approaches [Isermann 2005]. Knowledge based systems typically associates recognized behaviors with predefined known faults and hence, are less likely to detect an unknown fault. Model-based approaches are potentially very accurate. The correct behavior of each component is modeled analytically. The system output is compared to the modeled output and a high residual indicates a fault. The diagnosis can be deduced from the information the model provided. In complex autonomous systems, the task of modeling the behavior of components is very hard or even impossible. Data driven approaches are model free. The online data is usually used to statistically differentiate a fault from a normal behavior. However, this task may be not as light as the system requires.

In our previous work [Khalastchi *et al.*, 2013] we presented an unsupervised approach for fault detection in the domain of autonomous systems. This approach combines model based and data driven approaches, and shows a high rate of detection and a low rate of false positives. In this paper we aim to show a hybrid approach that uses an accurate **unsupervised** approach for fault detection similar to our previous approach, to create more accurate and light fault detection and diagnosis model in a **supervised** manner.

Empirical evaluation on simulated and real word domains show that the learnt fault detection model is more accurate than the original unsupervised approach. Finally, we argue that this hybrid approach can be generalized to a classification problem with an unlabeled training set.

2. Related work

Steinbauer conducted a survey on the nature of faults of autonomous robot systems [Steinbauer 2011]. The survey participants are developers competing in different leagues of the Robocup competition [Robocop]. The reported faults were categorized as hardware, software, algorithmic and interaction related faults. The survey concludes that hardware faults have a high negative impact on mission success. In this paper we focus on detecting such faults.

We presented an unsupervised approach for sensor fault detection that combines model-based and data driven techniques to achieve greater accuracy [Khalastchi *et al*, 2013]. The hybrid approach presented in this paper utilizes this unsupervised approach.

The chosen unsupervised approach has a high detection rate for single dimension faults such as "drift" and "stuck". These types of faults appear in a variety of related domains. For example, the Advanced Diagnostics and Prognostics Testbed [ADAPT] depicts the following faults to sensors on an electrical circuit. This testbed is used for the diagnosis competition [DXC,2011]. Another example is the work of [Hashimoto *et al.*, 2005] that uses kalman filters along with kinematical models to detect sensor faults such as "stuck",

"abrupt" and "scale" on a mobile robot. Our hybrid approach relies on a function that returns the state of the sensor (i.e. abrupt, drift, stuck, etc.).

[Leeke *et al,* 2011] present a methodology for generating efficient error detection mechanisms. Their approach relies on injecting faults into data that is used as a training set for a learning algorithm. The learnt error detection model is of high accuracy. Our hybrid approach is similar in concept, but does not rely on fault injection.

3. Decreasing the False Positive Rate

In this section we describe the problem of fault detection in the domain of unmanned vehicles. We continue with demonstrating how the unsupervised labeling is done. Finally, we describe the learning process.

3.1 Problem Description

Let $A = \{a_1 \dots a_n\}$ be a set of attributes that are monitored in real time (e.g. air-speed, heading, pitch, altimeter, etc.) and let $V_t = \{v_1 \dots v_n\}$ be the set of values for attributes $\{a_1 \dots a_n\}$ at time t where $v_i \in \mathbb{R}$ is the value assigned to a_i at time t. Past data of m time units of these values $H_m(t) = (V_{t-m}, V_{t-m+1}, V_{t-m+2}, \dots, V_t)$ is also available. $H_m(t)$ is a sliding window containing at time t the latest m values of the monitored attributes. In addition, unlabeled past recordings of the unmanned vehicle operations are also available. These recordings can be used as a training set for a machine learning algorithm if labeled. We denote this training set as $\mathcal{H} = \{H_{l_1}(e_1), H_{l_2}(e_2), \dots, H_{l_k}(e_k)\}$ where l_i is the length of operation i and $e_i$is the end time operation i, thus $H_{l_i}(e_i)$ denotes all the values recorded for the monitored attributes in A during operation i.

Given $A, V_t, H_{m(t)}$and $\mathcal{H}$, the goal is to online recognize whether a fault has occurred to any of the attributes in A. This decision should be made as quick as possible after a fault has occurred, and should be as accurate as possible. By 'accurate' we mean that a fault detector should have a high detection rate and a low false positive rate. In addition, a minimal set of diagnosis that includes the root cause of the fault should be returned.

3.2 The Outline of the Approach

We introduce a hybrid approach which consists of an **offline** preprocess and an online fault detection and diagnosis process. The offline preprocess conducts an **unsupervised** algorithm to label an unlabeled training set. Then, a **supervised** learning algorithm is used to construct a fault detection and diagnosis model that can be used **online** with greater accuracy. Figure 1 depicts the outline of our approach.

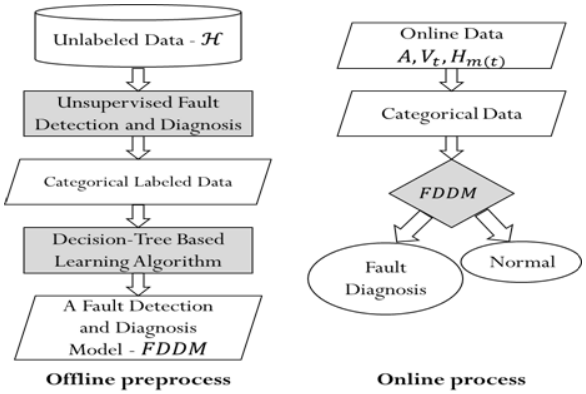


Figure 1: The Outline of the Approach

3.2.1 The offline preprocess

The unlabeled past operations recordings $\mathcal{H}$are delivered as an input to an unsupervised fault detection algorithm. This algorithm is described in section 3.3. In summary this algorithm translates the behavior of each attribute in each sliding window to normal or to one of pre-defined suspicious patterns (e.g. stuck, drift). In addition, each sliding window is labeled according to the unsupervised classification (i.e. "*Normal*", or the diagnosis of the fault). However, a small portion of the normal examples may be misclassified as faults; these are the false positives of the unsupervised fault detection. The resulted labeled data is fed into a decision-tree based leaning algorithm. The resulted fault detection and diagnosis model – $FDDM$ can be used online, and is more accurate than the original unsupervised fault detection and diagnosis approach as shown in the results section.

3.2.2 The online fault detection process

The online input is a time series confined into a sliding window $H_{m(t)}$. With each time step t the data in $H_{m(t)}$ is transformed into a categorical data which can be fed into the fault detection and diagnosis model - $FDDM$ (resulted by the offline preprocess). The fault detection and diagnosis model classifies the data presented in $H_{m(t)}$ as *normal* or as faulty by returning the fault diagnosis as a classification. In section 3.5 we describe the fault detection and diagnosis online process in detail.

3.3 Unsupervised Labeling

Each past operation of the autonomous system $H_{l_i}(e_i) \in \mathcal{H}$is fed into an unsupervised diagnosis engine. This diagnosis engine uses a heuristic decision which is based on a prior knowledge of a structural model as well as the online consumed data $H_{m(t)}$ (sliding window) to determine an occurrence of a fault and its diagnosis. We chose this approach due to its very low rate of false positives and the high fault detection rate which usually is 1 or very close to 1. Note that any unsupervised fault detection and diagnosis approach can be used to label the unlabeled training set.

Being online and unsupervised, the approach relies on correlated attributes to provide the necessary comparison between expected and unexpected behavior of attributes. The approach assumes that correlated attributes behave as redundant to one another. This is usually the case for the domain of unmanned vehicles with a rich array of sensors and modeled variables.

Each attribute is subjected to tests by suspicious pattern recognizers. A suspicious pattern recognizer is a part of the fault detection system input and is domain specific. The latest m values of an attribute extracted from $H_{m(t)}$ are

tested. For example, we use a *drift* test and a *stuck* test. However, when an attribute shows a suspicious pattern it does not necessarily suggest a fault; it could be a reaction to a normal action of the unmanned vehicle. For example, maintaining altitude may appear as *stuck*, and altitude climbing may appear as a *drift*.

To differentiate between a normal reaction and a fault, the approach uses a heuristic decision based on a structural model. A structural model depicts components dependency. The heuristic decision compares an attribute that shows a suspicious pattern with an attribute that used to be correlated to it in the previous sliding window. If they do not share component dependency and show different patterns then this is due to a fault. Otherwise, it might be a reaction to a normal action of the unmanned vehicle.

For example, if the altimeter is suspected for a *drift*, and the GPS indicated altitude is also drifting then this is probably due to the altitude climbing action of the UAV (and not a fault). If the altimeter is *stuck* while the GPS indicated altitude is not, then it is probably due to a fault. Furthermore, these attributes are dependent on different subsystems. This fact makes these attributes less likely to be affected by the same fault.

The data of each sliding window, $\forall t \in (m \dots e_i),\ H_{m(t)} \subseteq H_{l_i}(e_i) \in \mathcal{H}$, is transformed into a line of categorical data with the addition of the classification made by the unsupervised fault detection algorithm. A recognized suspicious pre-defined pattern is replaced by its class. We use a *drift* and a *stuck* pattern recognizers which correspond to "*drift*" and "*stuck*" categories. An additional "*regular*" category is given to an attribute which is not suspected by any suspicious pattern recognizer.

For example, assume a sliding window of size $m = 4$ containing the values of 3 attributes at a time step t:

Time step	a_1	a_2	a_3
t-4	0.9	1	3
t-3	0.1	2.5	3
t-2	0.05	3	3
t-1	0.15	3.1	3
t	0.05	3.9	3

Assume that the unsupervised diagnosis engine reported a fault for this time step including diagnosis components $\{c_3, c_{15}\} \subseteq \{c_1, \dots c_{40}\}$. The data of the sliding window will be registered as one training sample, where a_1 is categorized as "*regular*", a_2 is categorized as "*drift*", and a_3 is categorized as "*stuck*". In addition, this sample is classified by the diagnosis of the fault - $\{c_3, c_{15}\}$. Thus, the registered training sample for this time step is: *Regular,Drift,Stuck,* $\{\boldsymbol{c_3, c_{15}}\}$. Note that the fact that this sample contains *drift* and *stuck* does not necessarily entail a fault. In some cases it may indicate a normal behavior.

The "Fault" classification (e.g. $\{c_3, c_{15}\}$) may be correct or not; it depends on the accuracy level of the unsupervised approach. The next section will discuss how this inaccuracy affects the constructed decision tree.

3.4 A Decision Tree Based Fault Detection Model

3.4.1 The effect of falsely classified training examples

Decision trees have some tolerance towards falsely labeled examples in their training set. During the model construction, the algorithm grows a decision tree. In each step the algorithm computes the information gain obtained by selecting each attribute and chooses the one with the highest information gain ($IG(a_j)$). The information gain is determined by the entropy of the attribute which is affected by the ratio between the normal and faulty samples. The lower faulty samples the higher information gain. If a portion of samples are misclassified as "fault" then this might reduce the information gain of a node. However, if the degree of reduction is small enough the construction of the tree is unaffected.

The degree of information gain reduction due to falsely classified training examples is dependent on several factors. Let

- S be the training set.
- α_i be $|S_{a=v_i \wedge Class=fault}|$ the size of examples of which attribute a has the value v_i and the classification is "Fault" (i.e. some diagnosis classification of a fault) when all examples are classified correctly by an oracle.
- β_i be $|S_{a=v_i \wedge Class=normal}|$ the size of examples of which attribute a has the value v_i and the classification is "*Normal*" when all examples are classified correctly by an oracle.
- x_i be the number of samples from $|S_{a=v_i}|$, that were falsely classified as "Fault" due to false positives of the unsupervised approach (note that $|S_{a=v_i}| = \alpha_i + \beta_i$).

The effect of the misclassified x_i samples on the information gain of attribute a is the new (affected) information gain $IG_x(a)$ minus the original information gain $IG(a)$:

$$f(x) = IG_x(a) - IG(a)$$

$$f(x) = H(S) - \sum_{v_i} \frac{|S_{a=v_i}|}{|S|} H_x(S_{a=v_i}) - H(S) + \sum_{v_i} \frac{|S_{a=v_i}|}{|S|} H(S_{a=v_i})$$

Where H is the entropy function and H_x is the entropy affected by falsely classified examples. For given x_i falsely classified samples in $S_{a=v_i}$ the affected entropy is: $H_x(S_{a=v_i}) = -\frac{\alpha_i+x_i}{\alpha_i+\beta_i}\log\frac{\alpha_i+x_i}{\alpha_i+\beta_i} - \frac{\beta_i-x_i}{\alpha_i+\beta_i}\log\frac{\beta_i-x_i}{\alpha_i+\beta_i}$. Note that x_i samples that are falsely added to α_i are in the expense of β_i. After some algebra $f(x_i)$ can be presented for given x_i falsely classified examples in $S_{a=v_i}$ as:

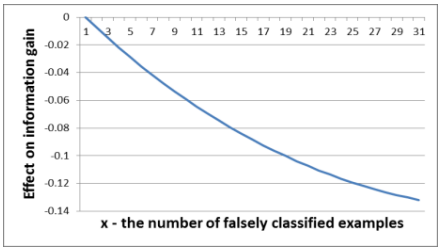


Figure 2: The negative effect on

$$f(x_i) = \frac{1}{|S|}\left(\alpha_i \log\frac{\alpha_i + x_i}{\alpha_i} + \beta_i \log\frac{\beta_i - x_i}{\beta_i} + x_i \log\frac{\alpha_i + x_i}{\beta_i - x_i}\right)$$

We can see that when $x_i = 0$ then $f(x_i) = 0$. When x_i grows, $f(x_i)$ decreases. This affect is depicted in Figure 2. Figure 2 illustrates an example for this effect on the information gain where $|S| = 300, \alpha_i = 20, \beta_i = 100$.

As x_i grows, the information gain decreases. The negative effect will eventually return to 0 as x_i grows. When $\alpha_i + x_i = \beta_i$ the number of samples for each classification is the same as the original, only with opposite classifications; the entropy is the same as the original entropy and thus the information gain is unaffected.

If the information gain of some attribute $IG(a_i)$ is greater than the information gain of another attribute $IG(a_j)$ when all training set examples are classified correctly by an oracle, then for a small enough number of falsely classified examples x the effect of $f(x)$ will not change the decision tree construction $IG_x(a_i) = IG(a_i) + f(x) > IG(a_j)$.

In some cases the number of falsely classified examples x will affect the decision tree construction $IG(a_i) + f(x) < IG(a_j)$. This might lead to the appearance of false positives when using the resulted fault detection and diagnosis model. When compared to a model that was constructed of correctly classified training examples this approach has more false positives due to the effect of $f(x)$. However, this approach has significant less false positives than the original unsupervised approach, as we show in the results section.

3.4.2 Using the *FDDM* online

When given a new online input, we consume it in a sliding window fashion $H_m(t)$. We transform the time-series data in $H_m(t)$ to a line of categorical data in the same manner we used in the original unsupervised approach. Only when the new line is different than the previous line (i.e. some attribute changed its state) the new line is fed into the offline-learnt fault detection and diagnosis model - $FDDM$. The $FDDM$ decides (online) whether or not it is a fault. In case of a fault the $FDDM$ returns a diagnosis as the classification.

For example, consider a static-system failure. One of the expressions of this failure is the frozen value of the altimeter. In $H_m(t)$ the values of the altimeter attribute are all equal, while the values of the GPS indicated altitude attribute diverse. The equal values of the altimeter are recognized as a suspicious pattern by the stuck-pattern detector. Therefore, the corresponding categorical line to $H_m(t)$ has the value "*stuck*" for the altimeter attribute and the value "*regular*" for the GPS indicated altitude attribute (other attributes also get their own categorical values) . This line is fed into the $FDDM$ that decides whether or not these values express a fault. It is possible that the next categorical line that corresponds to $H_m(t+1)$ will not be different than its predecessor. In this case, the $FDDM$ is not triggered again; only if at least one of the attributes changed its state (e.g. from regular to stuck as the altimeter) then the $FDDM$ is triggered.

To summarize our hybrid approach, in the first stage we use an unsupervised fault detection and diagnosis algorithm to label unlabeled training set. The suspicious pattern detectors of the unsupervised approach are used to transform the time-series data in each sliding window into categorical data. Each sample is classified according to the unsupervised decision (i.e. "*Normal*" or other fault modes). Then, we apply a decision tree based learning algorithm on the training data and produce a fault detection and diagnosis model - $FDDM$ that can be applied online. The online process uses the same suspicious pattern detectors to produce a categorical line for each sliding window. The line is fed into the $FDDM$ which was learnt offline. The $FDDM$ makes a choice whether or not the online input is an expression of a fault. In case of a fault, the classification is the diagnosis.

4. Experiment Setup

To examine our approach we tested the fault detection accuracy as well as the diagnosis accuracy. We use three domains to test the fault detection accuracy. The first is a high fidelity flight simulator [FlightGear] the second is a commercial UAV, and the third is a laboratory robot *Robtican1* [Robotican] (see Figure 3). In addition, we use the FlightGear domain to test the diagnosis accuracy. We expect our proposed hybrid approach to be more accurate in fault detection and diagnosis than the original unsupervised approach.

Flighgear domain: *Flighgear* is an open source flight simulator designed for research purpose and is used for a variety of research topics. FlightGear has built-in realistically simulated instrumental and system faults. For example, if the vacuum system fails, the HSI gyros spin down slowly with a corresponding degradation in response as well as a slowly increasing bias/error.

We recorded 32 flights. Each flight had duration of 5 minutes, and included a take-off and left and right turns. 23 attributes were sampled in 4Hz. Each flight was injected with a different type of fault. Each fault had duration of 35 seconds and was injected twice to the same flight at random times. In total, we tested 11 different types of instrumental and system failures. In total, the test set contains 25,977 examples out of which 5,880 are expressions of faults.

The unsupervised process applied on the training set achieved a detection rate of 1 (all faults were detected). The resulted categorical and classified data was used as a training set.

In addition, due to its richness of data and deep level of components dependencies, we use the FlightGear domain to further test the diagnosis accuracy of the proposed hybrid approach. We tested two types of system failures and 4 types of instrumental failures. In total, 6 flights were used as a training set and 12 flights were used as a testing set. The goal was to accurately detect and diagnose the faulty components out of 40 possible components.

Commercial UAV domain: The real UAV domain consists of 6 recorded real flights of a commercial UAV. 53

attributes were sampled in 10Hz. The attributes consists of telemetry, inertial, engine and servos data. Flights duration varies from 37 to 71 minutes. The UAV manufacture injected a synthetic fault to two of the flights. The first scenario is a value that drifts down to zero. The second scenario is a value that remains frozen (stuck). The detection of these two faults were challenging for the manufacture since in both scenarios the values are in normal range. These two flights were used as a test set. The remaining four flights were used as a training set where into two flights we injected similar synthetic faults. In total, the test set contains 65,741 examples out of which 1,593 are expression of faults.

Laboratory robot domain: *Robotican1* is a laboratory robot that has 2 wheels, 3 sonar range detectors in the front, and 3 infrared range detectors which are located right above the sonars, making the sonars and infrareds redundant systems to one another. This redundancy reflects real world domains such as unmanned vehicles. In addition, the Robotican has 5 degrees of freedom arm. Each joint is held by two electrical engines. These engines provide a sensed reading of the voltage applied by their action.

Figure 3: Robotican1

We devised 10 different scenarios that included different injected faults while the robot performed different tasks. Faults were injected to each type of sensor (motor voltage, infrared and sonar). The injected faults to the sensors were of type *stuck* or *drift*. These faults were injected to one or more sensors in different time intervals. 15 attributes were sampled in 8Hz. Scenarios duration lasted 10 seconds where the last 5 seconds expressed a fault. 4 scenarios were used as an unlabeled training set and the other 6 were used as a test set. Note that in this domain, the training set did not cover all the examples included in the test set.

For the supervised learning we have experimented with several decision tree algorithms: ID3, J48, and a Random Tree [Breiman, 2001]. As expected the Random Tree performed better and its results on the three domains are shown in the next section.

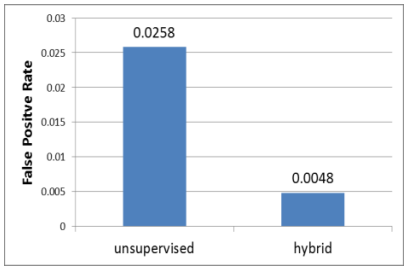


Figure 4: FP rate,unsupervised vs. hybrid

5. Results

Figure 4 illustrates the average false positive rate of the hybrid vs. the original unsupervised approach, taken over the 21 test flights of the FlightGear domain, using a sliding window size of 250. The hybrid approach significantly improved the false positive of the unsupervised algorithm.

To demonstrate the degree of reduction of the false positive rate by the suggested hybrid approach we used different sizes of sliding windows during the offline training phase. Smaller sizes create more opportunities for reports and thus more opportunities for false positives. Figure 5 illustrates the degree of reduction in the average false positive rate over the 21 test flights in the FlightGear domain. Note that the false positive rate is in logarithmic scale. We can see that with each size of sliding window the false positive rate of the hybrid approach is significantly lower than the original unsupervised approach.

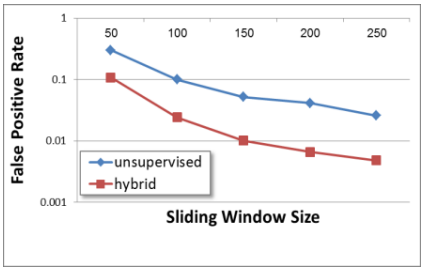


Figure 5: FP rate vs. s.window size

The different parameters used by the unsupervised approach during the offline phase can be viewed as different unsupervised approaches; each with its own rate of false positives. The hybrid approach contributes to the reduction of false positive rate for each of these unsupervised approaches.

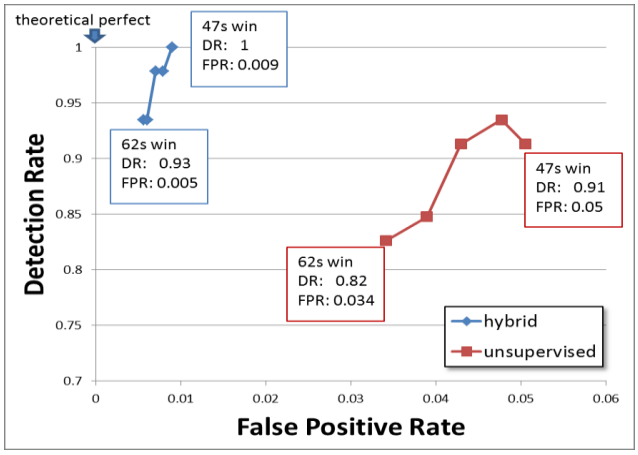


Figure 6: ROC - Hybrid vs. Unsupervised

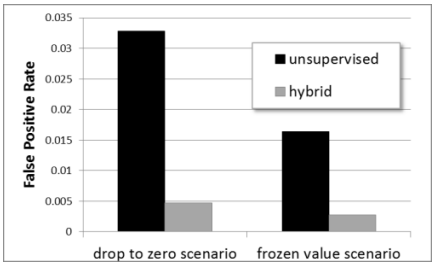


Figure 7: UAV, unsupervised vs. hybrid

Satisfied by the very low rate of false positives, we decreased the sliding window size used **online**. It is suggestible to use a smaller m for $H_m(t)$ when classifying an online input than the m used during the offline training. This increases the number of reports, and since the false alarm rate is very low, we can

tolerate an increase of false positives in return for a higher true positives rate.

Figure 6 illustrates the ROC of false alarm rates and the detection rates of the unsupervised approach verses the hybrid approach under the influence of a changing size of the online sliding window (62sec – 47sec). Note that scale of Figure 6 zooms in on high detection rate (close to 1) and low false alarm rate (close to 0). The added of false positives to the hybrid approach is of little significance while the effect on the unsupervised approach is apparent.

In addition, the detection rate of the hybrid approach is getting higher as the size of the sliding window decreases. This is explained by the fact that a smaller size of a sliding window increases the frequency of state changes and hence the total amount of reports. Therefore, there is a greater chance for detection as well as some false positives. The hybrid approach gets a lower rate of false alarms and a higher rate of fault detection than the original unsupervised approach.

In the UAV domain the hybrid approach keeps a similar trend. In the two examined scenarios, both the hybrid and unsupervised approaches had a detection rate of 1. However, the hybrid approach had a significantly lower false positive rate than the unsupervised approach as figure 7 shows.

In the *Robotican1* domain, even though the training set did not include all possible faults that were included in the test set, the detection rate of the learnt fault detection model was 1. Being online and unsupervised, it is not surprising that the unsupervised approach also scored a detection rate of 1 on the test set. However, it is interesting to note that the offline learnt *FDDM* of the hybrid was able to generalize the heuristic decision of the unsupervised approach such that unseen faults were detected.

The average false alarm rate of the unsupervised approach on the 6 tested scenarios was 0.067 while the hybrid approach scored 0.041. Again, the hybrid approach reduced the false positive rate.

We also tested the diagnosis of the proposed approach on the FlightGear domain as table 1 depicts.

The unsupervised approach produced very good results: A detection rate of 1, false alarm rate of 0.0086, and the diagnosis set contained an average of 2.88 components out of 40 possibilities, and always included the single root cause, making the diagnosis false positive rate as 0.048. But still, the hybrid approach is able to improve the results. The hybrid approach got a detection rate of 1, a false alarm rate of 0.0077, and an average diagnosis set size of 2.14 out of 40 possible components that included the single root cause, making the diagnosis false positive rate 0.029.

Table 1:diagnosis results, FlightGear domain

Approach	Fault Detection rate	False alarm rate	Diagnosis true positive rate	Diagnosis false positive rate
Unsupervised	1	0.0086	1	0.048
Hybrid	1	0.0077	1	0.029

6. Discussion

The offline step labels the data with an unsupervised approach. An alternative approach for labeling the data is a clustering algorithm (e.g. K-means where k=2). However, an unsupervised fault detection approach is more specific to the fault detection problem and thus expected to be more accurate than the general clustering algorithm.

We chose to demonstrate the hybrid approach with the use of our previous unsupervised approach [Khalastchi *et al.* 2013] since it showed a high detection rate and a very low false positive rate. Any other highly accurate unsupervised approach could have been used for that matter. The high detection rate is very important since all faults should be labeled as such.

The learnt *FDDM* generalized the original heuristic decision of the unsupervised approach. The model is independent of online correlation calculations and thus is lighter and less susceptible to false positives than the original unsupervised approach. Moreover, The *FDDM* returned less diagnosis candidates than the original unsupervised approach, further isolating the root cause possibilities. Finally, we argue that the hybrid approach could be generalized to any classification problem when the training data is unlabeled.

References

[ADAPT] http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/adapt-diagnostics/

[Breiman, 2001] Breiman L. "Random forests." *Machine learning 45.1 : 5-32.*

[DXC, 2011] International Diagnostic Competition - website, http://sites.google.com/site/dxcompetition2011/

[FlightGear] website, http://www.flightgear.org/

[Hashimoto, 2005] Hashimoto M. A multi-model based fault detection and diagnosis of internal sensors for mobile robot. *Intelligent Robots and Systems ,pp.3787- 3792.*

[Isermann 2005] Isermann R. Model-based fault-detection and diagnosis—Status and applications. *Annual Reviews in Control, 29(1), 71–85.*

[Khalastchi *et al*, 2013] Khalastchi E., Kalech M., Rokach L. Sensor fault detection and diagnosis for autonomous systems. In proceedings: The Twelfth International Conference on *Autonomous Agents and Multi-Agent Systems.*

[Leeke *et al,* 2011] Leeke M, Saima A, Arshad J, and Sarabjot S.A., A methodology for the generation of efficient error detection mechanisms." In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on, pp. 25-36.*

[Robocop] Robotcup competition, website, http://www.robocup.org/

[Robotican] website, http://www.robotican.net/

[Steinbauer 2011] Steinbauer G. a survey on the nature of faults of autonomous robot systems. website, http://www.ist.tugraz.at/rfs/index.php/Main_Page

Improving the Diagnostic Performance for Dynamic Systems by Using Conflict-Driven Model Decomposition*

Anibal Bregon[1], Alexander Feldman[2], Belarmino Pulido[1], Gregory Provan[2], Carlos Alonso-González[1]
[1] Depto. de Informática, Universidad de Valladolid, Spain
[2] Dept. of Computer Science, University College Cork, Ireland

Abstract

This work studies potential ways of integration of two techniques for fault detection, isolation, and identification in dynamic systems: the LYDIA-NG suite of diagnosis algorithms and the Consistency-based Diagnosis approach with Possible Conflicts. By integrating both techniques, LYDIA-NG will benefit from a more efficient fault detection and isolation task, and Possible Conflicts will benefit from the identification capabilities of LYDIA-NG. In this paper, we define a common framework that integrates both techniques, and then we apply the proposed integrated approach to a three-tank system, and draw some conclusions about potential ways of integration.

1 Introduction

The need for safety and reliability in engineering systems provides the motivation for developing Integrated Systems Health Management (ISHM) methodologies that include efficient fault diagnosis mechanisms. In this work we focus on model-based approaches to on-line fault diagnosis of dynamic systems. Online methods for model-based diagnosis require the use of quick and robust fault detection methods to establish discrepancies between observed and expected system behavior. However, accurate and timely online fault diagnosis of complex dynamic systems is difficult and can be computationally expensive [Isermann, 2006].

In this work we study how to combine two techniques suitable for model-based diagnosis of dynamic systems looking for better performance in on-line fault diagnosis. We have used the LYDIA-NG suite of algorithms [Feldman *et al.*, 2013]. The main idea of LYDIA-NG is to perform multiple simulations for various hypothesized health states of the plant. The output of these multiple simulations is then processed and combined into single diagnostic output. LYDIA-NG has been successfully used for complex applications like space satellites [Feldman *et al.*, 2013]. However, when applied to online fault diagnosis of large dynamic systems, running all the hypothesized health states becomes a quite difficult and time consuming task.

Several approaches have been proposed in recent years to deal with the complexity issue. System decomposition methods, have been proposed to reduce the complexity in the fault diagnosis task [Bregon *et al.*, 2012] by generating smaller simulation submodels which can run in parallel and provide independent diagnosis decisions. The Possible Conflict, PC, approach [Pulido and Alonso-González, 2004], is an off-line dependency compilation technique from the DX community, which decomposes the global system model into minimal submodels, and performs on-line behavior estimation using simulation, dynamic bayesian networks, or state-based neural networks [Pulido *et al.*, 2012]. If a discrepancy is found, a set of fault candidates is generated by a minimal hitting-set algorithm of the triggered PCs. However, additional techniques must be used to refine the set of fault candidates.

The goal of this work consists of integrating PCs within the LYDIA-NG diagnosis framework. First, PCs will decompose the global simulation model into a set of smaller simulation submodels. Then, PCs will be used for efficient online fault detection and fault localization, providing a subset of fault candidates from the minimal hitting-set of the fault parameters linked to the set of equations in the PC. The subset of fault candidates is then used as input to LYDIA-NG, where simulations are run only for each one of the fault candidates, and its result is processed and combined to provide the diagnosis output. The approach has been tested by using a three-tank system case study.

The rest of the paper is organized as follows. Section 2 presents the basic definitions and running example used in this work. Section 3 briefly introduces LYDIA-NG and PCs. Section 4 presents our proposal to integrate PCs within the LYDIA-NG diagnosis framework. Section 5 describes the experimental results obtained for the three-tank system. Section 6 presents related work. And, finally, section 7 presents the discussion and conclusions.

2 Concepts and Definitions

In this section we present our basic definitions and a running example that we use to illustrate the significant concepts of this paper. Since both LYDIA-NG and PCs are model-based diagnosis approaches, we provide a set of definitions about models and faults that will allow us to explain later both techniques using the same framework.

2.1 Definitions

For the purpose of this work we focus our description on continuous systems, with only one nominal state, and whose behavior can be described as a set Σ of Ordinary Differential Equations (ODEs). The model of our system will be the basic system description to perform diagnosis:

*A. Bregon, B. Pulido, and C. Alonso's funding for this work was provided by the Spanish MCI TIN2009-11326 grant.

Definition 1 (Model)**.** The system model is defined as $M(\Sigma, U, Y, X, \Theta)$, where: Σ is a set of ODEs, defined over a collection of known and unknown variables: U is a set of inputs, Y a set of outputs, X a set of state and intermediate, i.e. unknown, variables, and Θ is the set of parameters[1].

Definition 2 (System Description, SD)**.** SD is made up of (M, H, σ, Π), where

- H is the health-vector defined by means of $h_i \mid 1 \leq i \leq k$ health variables, that allow us to characterize the set of states in the system, i.e. each $h_i \in H$ is a potential mode for the system, either nominal or faulty.
- σ is a mapping function: $\sigma(M, H_c) \rightarrow M_{Hc}(\Sigma_{Hc}, U_{Hc}, Y_{Hc}, X_{Hc}, \Theta_{Hc})$, that given the model, M, and the current health status, Hc, provides the model for behavior estimation for the current mode (or current system description M_{Hc}): $\Sigma_{Hc} \subseteq \Sigma$, $U_{Hc} \subseteq U$, $Y_{Hc} \subseteq Y$, $X_{Hc} \subseteq X$, and $\Theta_{Hc} \subseteq \Theta$).
- Π is a mapping function $\Pi(\theta_{cc}) \rightarrow \{Hc \mid Hc \subseteq H\}$ that, given a set of parameters, provides the set of health variables that relate to the set of model parameters: $\theta_{cc} \subseteq \Theta_{cc}$.

An implicit assumption in our modeling approach is that we can use the same set of equations for both the nominal behavior estimation and the faulty behavior estimation, just changing the value of θ_i.

In model-based diagnosis, the model of the system is used to compute a residual signal, which is used for fault detection and isolation purposes. A residual is computed as the difference between the observed behavior (obtained via sensor outputs y_i) and the expected behavior (estimated by the system model, $\hat{y}_i$), and it is formally defined as follows:

Definition 3 (Residual)**.** A residual is a real-valued measure $R(y_i, \hat{y}_i)$ of the difference between real and simulated system output at time t.

2.2 Running Example

In this paper, we use the three-tank system shown in Fig. 1 as the running example. The three tanks are denoted as T_1, T_2, and T_3. They all have the same area $A_1 = A_2 = A_3 = 3$ [m^2]. The experiments are performed assuming the gravity $g = 10$ and the liquid with density $\rho = 1$.

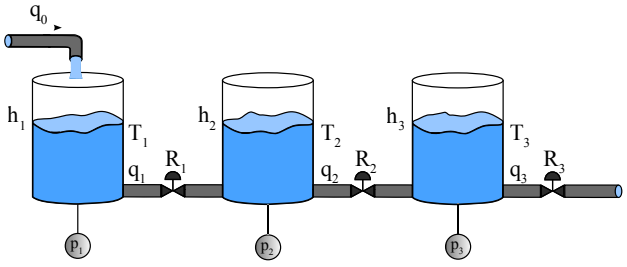


Figure 1: Diagram of the three-tank system.

Tank T_1 gets filled from a pipe q_0 with a constant flow of 1.5 [m^3/s]. It drains into T_2 via a pipe q_1. The liquid level is denoted as h_1. There is a pressure sensor p_1 connected to T_1 that measures the pressure in Pascals [Pa]. Starting from the Newton's (and Bernouli's) equations and manipulating them we derive the following Ordinary Differential Equation (ODE) that gives the level of the liquid in T_1:

$$\frac{dh_1}{dt} = \frac{q_0 - k_1\sqrt{h_1 - h_2}}{A_1} \tag{1}$$

In Eq. 1, the coefficient k_1 is used to model the area of the drainage hole and its friction factor. We emphasize the use of k_1 because, later, we will be "diagnosing" our system in term of changes in k_1. Consider a physical valve R_1 between T_1 and T_2 that constraints the flow between the two tanks. We can say that the valve changes proportionally the cross-sectional drainage area of q_1 and hence k_1. The diagnostic task will be to compute the true value of k_1, given p_1, and from k_1 we can compute the actual position of the valve R_1. The water levels of T_2 and T_3, denoted as h_2 and h_3 respectively, are given by:

$$\frac{dh_2}{dt} = \frac{k_1\sqrt{h_1 - h_2} - k_2\sqrt{h_2 - h_3}}{A_2}, \tag{2}$$

$$\frac{dh_3}{dt} = \frac{k_2\sqrt{h_2 - h_3} - k_3\sqrt{h_3}}{A_3}. \tag{3}$$

Values k_1, k_2, and k_3, are constant values with no physical meaning, and we have set them with a value of 0.75. Finally, we turn the water level into pressure:

$$p_i = \frac{g\, h_i\, A_i}{A_i} = g\, h_i \tag{4}$$

where i is the tank index ($i \in \{1, 2, 3\}$). To observe the behavior of the system we have an observational model, that allows us to know or read each value p_i. We use p_i^* to distinguish the measured variable from the model output p_i as $p_i^* = p_i$.

It is assumed that the initial water level in the three tanks is zero. Additionally, we make explicit the relation between the state variables, h_i in our example, and their derivatives, dh_i, as $h_i = \int dh_i \cdot dt$. These equations allow us to select an integral or differential approach for behavior simulation, depending on the selected causality. These equations make no influence in the diagnosis results, because they will have no θ_i, and consequently no health status.

3 Algorithms

This section presents the fundamental ideas of the LYDIA-NG diagnosis framework and the structural model decomposition approach with PCs.

3.1 LYDIA-NG

The basic idea of the LYDIA-NG diagnostic library is to perform multiple simulations for various hypothesized health states of the plant. The output of these simulations is then processed and combined into single diagnostic output.

The LYDIA-NG diagnostic library consists of the following building blocks: (i) Generator of Diagnostic Assumptions; (ii) Simulation Engine; (iii) Residual Analysis Engine; (iv) Candidate Selection Algorithm; (v) System State Estimation Algorithm. Detailed description of these blocks can be found in [Feldman *et al.*, 2013].

Algorithm 1 shows the top-level diagnostic process. The inputs to Algorithm 1 are a model and a scenario, and the

[1]Since we are dealing with fault diagnosis, in our model we are mainly interested in every parameter suitable to model faulty behavior.

result is a diagnosis. Algorithm 1 supports a large variety of simulation methods that may or may not use time as an independent variable. The only requirement toward the simulation engine is to predict a number of variables whose types can be mapped to LYDIA-NG and to be relatively fast.

Algorithm 1 Diagnosis framework

```
1: function DIAGNOSE(SCN) returns a diagnosis
       inputs: SCN, diagnostic scenario
       local variables: h, FDI vector, health assignment
                        p, real vector, prediction
                        Ω, a set of diagnostic candidates
                        DIAG, diagnosis, result
2:     while h ← NEXTHEALTHASSIGNMENT() do
3:         p ← SIMULATE(M, γ, h)
4:         r ← COMPUTERESIDUAL(p, α)
5:         Ω ← Ω ∪ ⟨h, r⟩
6:     end while
7:     DIAG ← COMBINECANDIDATES(Ω)
8:     return DIAG
9: end function
```

The basic idea of Algorithm 1 is to simulate for various health assignments and to compare the predictions with the observed sensor data (i.e., telemetry). There are several important aspects of these algorithms that ultimately affect the diagnostic accuracy as measured by various performance metrics.

The first algorithmic property that determines many of the diagnostic performances is the order in which health-assignments are generated. In Algorithm 1 this is implemented in the NEXTHEALTHASSIGNMENT function. The latter subroutine also determines when to stop the search and should be properly parametrized depending on the model and the user requirements. In the standard LYDIA-NG diagnostic library we provide the breadth-first search (BFS), the depth-first search (DFS), and the backwards greedy stochastic search (BGSS) diagnostic search policies.

Each simulation produces a set of predicted values for a given health-assignment. The second important property of Algorithm 1 is the comparison and ordering of the diagnostic candidates. This is done by mapping the predicted and observed variables into a single real-number, called a *residual*.

Residual generation functions in LYDIA-NG bear resemblance to loss functions in decision theory. For example, residuals may be squared or absolute residuals [Feldman *et al.*, 2013]. A disadvantage of the squared residuals function is that it adds a lot weight to outliers.

3.2 Consistency-based diagnosis with PCs

In this section we present the fundamental ideas of Consistency-based Diagnosis and Possible Conflicts.

Consistency-based Diagnosis

Consistency Based Diagnosis (CBD) performs fault detection and fault isolation using only models of correct behavior in a *two stage process*. First, we identify if there exists a discrepancy between the observed behavior and the expected behavior, thus defining a discrepancy in terms of a residual. Corresponding to each residual, or discrepancy, is a conflict [Reiter, 1987]. Hence, fault detection consists of computing every conflict.

The second step is fault isolation, which consists of computing the minimal hitting sets of the conflicts, since they characterize the whole set of minimal diagnoses [Reiter, 1987]. Intuitively, a conflict is a set of components that cannot behave properly simultaneously, given the system description and current observations of abnormal behavior. In this work we use the Possible Conflicts (PCs) approach to avoid the on-line computation of conflicts and speed up overall fault isolation. PCs are designed to compute off-line those subsystems capable of becoming conflicts online.

For consistency-based diagnosis using PCs, we only use $\sigma(M, Hn)$ with Hn corresponding to a nominal mode. Since we are dealing with a continuous system working in one nominal mode, we can compute offline the set of PCs for $M_{Hn}(\Sigma_{Hn}, U_{Hn}, Y_{Hn}, X_{Hn}, \Theta_{Hn})$, as will be described later. The output of the consistency-based diagnosis using PCs is a set of fault candidates C defined in the lattice provided by Θ^*.

Model decomposition with PCs

The Possible Conflicts (PCs) approach [Pulido and Alonso-González, 2004] is a model decomposition method that finds (off-line) every subset of equations capable of generating conflicts. PCs provide the structural and causal model of a subsystem with minimal redundancy. The set of equations in a PC can be used to simulate the correct behavior of the subsystem. Hence, PCs can be used in CBD of dynamic systems [Pulido *et al.*, 2001]. For the sake of self-containment, we summarize here the proposal for PCs computation given in [Pulido and Alonso-González, 2004].

To compute PCs, we need the structural model of the system under study, which can be obtained from the set of equations in the system description, once we select a given working mode, tailored for our new problem formulation, instead of the original process which was suitable for system descriptions provided as hypergraphs [Pulido and Alonso-González, 2004]. We will illustrate the process using the three-tank system in Fig. 1, and the set of equations in its model as described in Section 2.2.

We need an abstraction of our model description $SD = (M, H, \sigma, \Pi)$. Let's assume we compute the set of PCs for a given nominal mode characterized by H_n. Using $\sigma(M, H_n)$, we obtain $M_{H_n} = (\Sigma_{H_n}, U_{H_n}, Y_{H_n}, X_{H_n}, \Theta_{H_n})$. For the structural model, we only need the information about the measured and unknown variables in each model equation. Thus each equation $\sigma_i \in \Sigma_{H_n}$ will provide one structural constraint $\sigma_i \rightarrow (S_i,\ X_i)$, where S_i accounts for the measured variables from U_{H_n}, Y_{H_n} in σ_i, and X_i accounts for the unknown (state or intermediate variables in σ_i).

For the three-tank system the structural model is made up of the following constraints:

$Constraint$	$Sensors$	$Unknowns$
c_1	$\{q_0\}$	$\{d_{h1}, h_1, h_2\}$
c_2	$\{\}$	$\{d_{h2}, h_1, h_2, h_3\}$
c_3	$\{\}$	$\{d_{h3}, h_2, h_3\}$
c_4	$\{\}$	$\{p_1, h_1\}$
c_5	$\{\}$	$\{p_2, h_2\}$
c_6	$\{\}$	$\{p_3, h_3\}$
c_7	$\{p_1^*\}$	$\{p_1\}$
c_8	$\{p_2^*\}$	$\{p_2\}$
c_9	$\{p_3^*\}$	$\{p_3\}$
c_{10}	$\{\}$	$\{h_1, d_{h1}\}$
c_{11}	$\{\}$	$\{h_2, d_{h2}\}$
c_{12}	$\{\}$	$\{h_3, d_{h3}\}$

where constraints c_1 to c_3 are related to equations (1) to (3); constraints c_4 to c_6 are related to the equation (4) for each one of the tanks; constraints c_7 to c_9 make explicit the diagnosis observational model, relating the output variable p_i and its associated sensor p_i^*; and constraints c_{10} to c_{12} make explicit the dynamic in the system: relation between the state variable and its derivative.

The first step in PC computation is to look for the complete set of minimally redundant subsets of equations, which we call the *Minimal Evaluation Chains* (MECs). A MEC represents a strictly overdetermined[2] set of equations that can potentially be solved using local propagation (elimination method): each MEC will have n constraints and $n-1$ unknowns. A summary of the algorithms used to compute MECs in a system can be found in [Pulido and Alonso-González, 2004]. The set of MECs in the system in Fig. 1 is:

- $mec_1 = \{c_7, c_4, c_{10}, c_1, c_5, c_8\}$
- $mec_2 = \{c_8, c_5, c_{11}, c_2, c_4, c_6, c_7, c_9\}$
- $mec_3 = \{c_9, c_6, c_{12}, c_3, c_5, c_8\}$

We need to know the different ways an equation can be solved, because we can deal with non-linear models. These ways are usually called the set of possible causal assignments for the variables in an equation. We assume that the set of possible causal assignments is known for the system model, and we build the complete set of valid causal assignments for the set of MECs, using exhaustive search [Pulido and Alonso-González, 2004]. We call each valid causal assignment *Minimal Evaluation Model* (MEM).

For the three-tank system, we assume that the causality is given by the expression in equations (1) to (6), except for the observational model (in this case we allow solving constraints ec_7 to ec_9 in both directions because we need to convert some system measurements Y in MEM inputs, U_{pc}). The set of MEMs for the three-tank system and their discrepancy nodes are shown in Table 1.

Table 1: MEMs for the three-tank system and their discrepancy nodes.

MEM	Discrepancy	Parameters
$\{c_7, c_4, c_{10}, c_1, c_5, c_8\}$	p_1^*	k_1, A_1
$\{c_8, c_5, c_{11}, c_2, c_4, c_6, c_7, c_9\}$	p_2^*	k_1, k_2, A_2
$\{c_9, c_6, c_{12}, c_3, c_5, c_8\}$	p_3^*	k_2, k_3, A_3

Fault detection and isolation using PCs

In the MEM there is a special node called *discrepancy* node (representing the only variable that is estimated by two different ways). Therefore, that node is the potential source of a residual, or discrepancy, using only the values of measured variables as inputs, and the past value of state-variables.

In CBD [Reiter, 1987; de Kleer and Williams, 1987] a conflict arises given a discrepancy between observed and predicted values for a variable. Under fault conditions, conflicts are observed when the model described by a MEM is evaluated with available observations and produce a discrepancy, because the model equations and the input/measured values are inconsistent [Reiter, 1987; de Kleer and Williams, 1987]. This notion of possible discrepancy generation leads to the definition of *Possible Conflict*:

Definition 4 (Possible Conflict)**.** The set of constraints in a MEC that give rise to at least one MEM.

Every MEC in the three-tank system has one MEM. Hence, we have three PCs. Each MEM is the computational model for a PC, and each equation in a MEM contains zero or more parameters that can be the source of potential faults (θ_{cc} in our model description). The set of parameters related to each PC is also shown in the fourth column in Table 1. Given a non-zero residual, we then isolate the fault parameters involved in the *pc* structural model: Θ_{pc}. This information is the basis for the integration of Consistency-based diagnosis of dynamic systems with Possible Conflicts and LYDIA-NG.

4 On-line Fault diagnosis with LYDIA-NG and PCs

In CBD, diagnosis must discriminate among 2^N behavioral mode assignments when just correct, $ok(\cdot)$, and incorrect modes, $\neg ok(\cdot)$, are present for N components. When B behavioral models are allowed, diagnosis must discriminate among B^N mode assignments. This is the problem faced by any model-based diagnosis proposal which attempts fault identification [Dressler, 1996], and it is also present in LYDIA-NG. In this section, we present an integration proposal, where the system model is partitioned using PCs. As explained in Section 2, the output of the consistency-based diagnosis using PCs is a set of fault candidates C defined in the lattice provided by Θ^*. Then, this set of diagnosis candidates is used as input to LYDIA-NG, thus reducing the number of health state simulations that needs to be considered by LYDIA-NG.

In our integration proposal, the simulation model for each PC uses some of the system measurements as input, and provides an estimation for exactly one variable (the potential discrepancy). Then, an executable model, SD_{pc_i} for each pc_i, is built. This executable model can be a simulation model, a state observer, or even a neural network [Pulido *et al.*, 2012]. Summarizing, the integration of LYDIA-NG and CBD with PCs is possible given the set of candidates, C: each candidate C_i is a subset of Θ_{pc}. Then invoking $\Pi(C_i)$, LYDIA-NG can obtain the set of health variables, Hc related to C_i, and use them as input for its search. Given the current implementation of LYDIA-NG, we can obtain the system description (system model) imposed by $Hc : \sigma(M, Hc)$, which is enough to characterize the current model and perform simulation of the Hc health status.

Algorithm 2 shows the algorithm for our integrated diagnosis framework. Y_{pc_i} denotes the set of input observations available for the executable model of a PC, SD_{pc_i}; and $\hat{Y}_{pc_i}$ represents the set of predictions obtained from SD_{pc_i}. The function OBTAINOBSERVATIONS obtains from the diagnostic scenario the observations which have to be used as input for each PC. Function ESTIMATEBEHAVIOR provides an estimation of a measured variable by using the executable model of each PC (either a simulation model, a state observer model, or a neural network).

For the detection part, to determine significant deviations from the PC residuals (PC residuals are computed by using an absolute residual function). We use the Z-test for robust fault detection using a set of sliding windows as detailed

[2] A redundant set of equations would be an Evaluation Chain. Since we are interested only on minimal conflicts, we just focus on the set of MECs that are by definition minimally overdetermined.

in [Daigle *et al.*, 2010]. A small window, N_2, is used to estimate the current mean of the residual signal, μ_r. The variance of the nominal residual signal is computed using a large window N_1 preceding N_2, by a buffer N_{delay}, which ensures that N_1 does not contain any samples after fault occurrence. The variance and the confidence level determined by the user are then used to dynamically compute the detection thresholds ϵ_r^- and ϵ_r^+.

Algorithm 2 Integrated PCs and LYDIA-NG diagnosis approach.

```
1: function PCs-LYDIA-DIAGNOSIS(SCN) returns a diagnosis
       inputs: SCN, diagnostic scenario
       local variables: Y_pc_i, set of input observations
                        Ŷ_pc_i, estimation from the PC
                        Θ_pc_i, fault parameters involved in the PC
                        h, FDI vector, health assignment
                        p, real vector, prediction
                        Ω, a set of diagnostic candidates
                        DIAG, diagnosis, result
2:     repeat
3:         Y_pc_i ← OBTAINOBSERVATIONS(SCN)
4:         Ŷ_pc_i ← ESTIMATEBEHAVIOR(SD_pc_i, Y_pc_i)
5:         r_pc_i ← COMPUTERESIDUALPC(Ŷ_pc_i, Y_pc_i)
6:         if Z-TEST(r_pc_i) < ε_r^- or Z-TEST(r_pc_i) > ε_r^+ then
7:             Θ_pc_i = confirm pc_i as a real conflict
8:             C ← MHS(C, Θ_pc_i)
9:         end if
10:    until Every pc_i is activated or time elapsed or a unique
       fault candidate has been isolated
11:    while h ← NEXTHEALTHASSIGNMENT(Π, C) do
12:        p ← SIMULATE(M, γ, h)
13:        r ← COMPUTERESIDUALLYDIA(p, α)
14:        Ω ← Ω ∪ ⟨h, r⟩
15:    end while
16:    DIAG ← COMBINECANDIDATES(Ω)
17:    return DIAG
18: end function
```

Once the initial set of fault candidates has been isolated, the LYDIA-NG part of the algorithm is run (as shown in algorithm 2). The algorithm takes the set of isolated fault candidates as input, and the NEXTHEALTHASSIGNMENT function only considers the health assignments related to the fault candidates. In this version of the integrated framework, the global system model is used as the simulation model, instead of the PC submodels, thus providing a more direct way to integrate both approaches. In future versions, the PC submodels will also be used as the simulation model in LYDIA-NG, thus providing faster simulation results.

5 Results

In this section we show some diagnosis results for our integrated framework. We first discuss the nominal situation, then, we present an on-line fault diagnosis scenario for a particular fault in the three-tank system and discuss the results obtained.

5.1 Nominal Scenario

For the nominal scenario, none of the three PCs found for the system is triggered. The advantage of including PCs within the LYDIA-NG framework is evident in this case. Since none of the PCs is triggered, LYDIA-NG is not run, thus avoiding the time-consuming simulations for the different health states when no actual fault has occurred in the system.

5.2 On-line fault diagnosis

This section briefly describes how LYDIA-NG runs with and without the use of PCs. The first phase is residual analysis, where LYDIA-NG runs a set of simulations such that a residual is computed for each simulation. Because LYDIA-NG uses real-value health variables, the space of potential diagnostic assumptions, and the corresponding set of simulations, is enormous, and infinite in the worst case. The heuristics used for generation of diagnostic assumptions are critical to the success and efficiency of LYDIA-NG.

LYDIA-NG ranks the residual outputs, discarding those candidates whose residual value is larger than the residual of the "all nominal" candidate. The remaining candidates are assigned probabilities of occurrence, using a method described in [Feldman *et al.*, 2013]. The fault isolation process assigns probabilities of failure to system components, and these are reported as ranked diagnoses.

In the following we compare the results for running LYDIA-NG with and without PCs. Without PCs, LYDIA-NG uses the global system model described earlier; with PCs (i.e., using Algorithm 2), the generation of diagnostic assumptions is governed by the PC-based algorithm.

For a diagnosis scenario with a 40% blockage fault in valve R_1 occurring at time 100 s, our results are as follows.

Non-PC-based Approach: LYDIA-NG computes residuals based on the difference between the pressures. The non-zero residual at time 104 s creates a set of simulations in which LYDIA-NG analyzes several valve %-blockage cases for R_1, R_2 and R_3. LYDIA-NG estimates the valve positions by "guessing" the *true* valve positions and computes the health probability by subtracting the commanded valve position from the estimated one. LYDIA-NG is able to isolate the most-likely fault as (R_1, 40%).

PC-based Approach: When computing diagnoses for this fault, at time 104 s, an increase in the residual of PC_2 is detected, and consequently k_1, k_2, and A_2 are selected as the initial set of fault candidates. At the next time step, at time 105 s, PC_1 is triggered, thus selecting k_1 and A_1 as possible fault candidates. A minimal hitting set algorithm is run, determining that the only single fault candidate in the system is k_1. At this point, the fault identification for k_1 is triggered by using LYDIA-NG.

Running this diagnosis scenario with a (trivial) input of R_1 (as derived from the candidate k_1), as opposed to R_1, R_2 and R_3, results in an 80× speedup of LYDIA-NG as compared to the non-PC approach. This is a result of reducing the diagnosis assumption space.

6 Related work

LYDIA-NG belongs to a class of MBD methods that use continuous-valued models and sensor data, and use entropy based methods for test selection to disambiguate diagnoses. It is a generalization of LYDIA-NG, which used discrete-value models.

In terms of diagnostics solvers, LYDIA-NG is related to the HyDE (Hybrid Diagnosis Engine) solver [Narasimhan and Brownston, 2007]. Another solver, FACT [Daigle *et al.*, 2010], can also use continuous-valued models and sensor data, but requires that the model be represented as a hybrid

bond graph. Given an anomaly, FACT first uses an observer-based approach (adopted from the FDI community) with statistical techniques for robust fault detection.

Recent works have demonstrated the similarities between model-based diagnosis approaches from the DX and the FDI communities [Cordier *et al.*, 2004]. In such framework, it has been demonstrated the equivalence of several structural model decomposition techniques such as PCs, minimal ARRs and Minimally Structurally Overdetermined sets [Armengol *et al.*, 2009]. As a consequence, the proposal in this work can be easily extended to other structural methods.

Using CBD we need to generate the set of candidates C and wait for every PC to be confirmed. An FDI approach would use exoneration using the structural information in the set of PCs. In CBD we wait for additional observations in order to reject modes that are not consistent with available information. Combining our results with LYDIA-NG provides an additional boost for candidate discrimination by including fault models through health variables.

7 Conclusions

This work has presented an integrated framework for on line fault detection, isolation and identification of dynamic systems.

Two different approaches have been integrated: The LYDIA-NG suite of diagnosis algorithms and the PCs framework for on-line CBD. LYDIA-NG is a simulation based diagnosis system that filters out diagnosis candidates discarding those of them that generates residuals larger than the *all-nominal* assumption, i.e., fault free and nominal system configuration. Although the system incorporates important facilities, such as diagnostic test generation based on entropy measure, its main drawback is the lack of focus for the initial set of candidates, which may be large, and the cost of simulating the complete system for every considered candidate. On the contrary, the set of PCs identifies minimal computational subsystems that decompose the complete system and that can be simulated independently. PCs are based on Reiter's theory of diagnosis from first principles and are able to generate fault isolation candidates from model of correct behavior without hypothesizing an initial set of candidates. Hence, using consistency-based diagnosis with PCs candidate generation is rather efficient, although additional techniques are required to further refine fault candidates for fault isolation and identification. They also lack some of the facilities incorporated in LYDIA-NG, like generation of diagnostic tests.

Our three-tank system running example shows the potential of this approach. First, when the system is fault free, no PC becomes a real conflict and no candidate is generated. This avoids running LYDIA-NG for fault detection, which is performed by the PCs approach, thus potentially providing a significant saving on computing time, depending on the size of the complete system and on the number and overlapping degree of the PCs. Second, when a fault is detected, PCs may generate a low number of fault candidates, depending on the number of PCs and its overlapping degree but also on the real faulty parameter, thus providing an automatic focus for LYDIA-NG fault candidate search.

References

[Armengol *et al.*, 2009] J. Armengol, A. Bregon, T. Escobet, E. Gelso, M. Krysander, M. Nyberg, X. Olive, B. Pulido, and L. Travé-Massuyès. Minimal Structurally Overdetermined sets for residual generation: A comparison of alternative approaches. In *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, SAFEPROCESS09*, pages 1480–1485, Barcelona, Spain, 2009.

[Bregon *et al.*, 2012] A. Bregon, G. Biswas, and B. Pulido. A Decomposition Method for Nonlinear Parameter Estimation in TRANSCEND. *IEEE Trans. Syst. Man. Cy. Part A*, 42(3):751–763, 2012.

[Cordier *et al.*, 2004] M.O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. Conflicts versus Analytical Redundancy Relations: a comparative analysis of the Model-based Diagnosis approach from the Artificial Intelligence and Automatic Control perspectives. *IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics*, 34(5):2163–2177, 2004.

[Daigle *et al.*, 2010] M. Daigle, I. Roychoudhury, G. Biswas, X. Koutsoukos, A. Patterson-Hine, and S. Poll. A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems. *IEEE Transactions of Systems, Man, and Cybernetics, Part A*, 4(5):917–931, September 2010.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligente*, 32:97–130, 1987.

[Dressler, 1996] O. Dressler. On-line diagnosis and monitoring of dynamic systems based on qualitative models and dependency-recording diagnosis engines. In *Proceedings of the Twelfth European Conference on Artificial Intelligence, ECAI-96*, pages 461–465, 1996.

[Feldman *et al.*, 2013] Alexander Feldman, Helena Vicente de Castro, Arjan van Gemund, and Gregory Provan. Model-based diagnostic decision-support system for satellites. In *Aerospace Conference, 2013 IEEE*, pages 1–14. IEEE, 2013.

[Isermann, 2006] R. Isermann. *Fault-Diagnosis Systems. An Introduction from Fault Detection to Fault Tolerance*. Springer, 2006.

[Narasimhan and Brownston, 2007] S. Narasimhan and L. Brownston. Hyde–a general framework for stochastic and hybrid model-based diagnosis. In *Proc. 18th International Workshop on Principles of Diagnosis (DX07), Nashville, USA*, pages 162–169. Citeseer, 2007.

[Pulido and Alonso-González, 2004] B. Pulido and C. Alonso-González. Possible Conflicts: a compilation technique for consistency-based diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics. Part B: Cybernetics*, 34(5):2192–2206, Octubre 2004.

[Pulido *et al.*, 2001] B. Pulido, C. Alonso, and F. Acebes. Lessons learned from diagnosing dynamic systems using possible conflicts and quantitative models. In *Engineering of Intelligent Systems. XIV Conf. IEA/AIE-2001*, volume 2070 of *LNAI*, pages 135–144, Budapest, Hungary, 2001.

[Pulido *et al.*, 2012] B. Pulido, J.M. Zamarreño, A. Merino, and A. Bregon. Using structural decomposition methods to design gray-box models for fault diagnosis of complex systems: a beet sugar factory case study. In A. Bregon and A. Saxena, editors, *Procs. of the First European Conference of the Prognostics and Health Management Society*, pages 225–238, Dresden, Germany, July 2012. www.phmsociety.org.

[Reiter, 1987] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–95, 1987.

Model-Based Diagnostic Decision-Support System for Satellites

Alexander Feldman[1] and **Helena Vicente de Castro**[2] and
Arjan van Gemund[3] and **Gregory Provan**[1]
[1]University College Cork, Cork, Ireland, e-mail: a.feldman@ucc.ie, g.provan@cs.ucc.ie
[2]CGI, Rotterdam, The Netherlands, e-mail: helena.vicente.de.castro@cgi.com
Delft University of Technology, The Netherlands, e-mail: a.j.c.vangemund@tudelft.nl

Abstract

We propose a novel framework for Model-Based Diagnosis (MBD) that uses active testing to decrease the diagnostic uncertainty. This framework is called LYDIA-NG and combines several diagnostic, simulation, and active-testing algorithms. We have illustrated the workings of LYDIA-NG by building a LYDIA-NG-based decision support system for the Gravity field and steady-state Ocean Circulation Explorer (GOCE) satellite. This paper discusses a model of the GOCE Electrical Power System (EPS), the algorithms for diagnosis and disambiguation, and the experiments performed with a number of diagnostic scenarios. Our experiments produced no false positive scenarios, no false negative scenarios, the average number of classification errors per scenario is 1.25, and the fault detection time is equal to the computation time. We have further computed an average fault uncertainty of 2.06×10^{-3} which can be automatically reduced to 9.5×10^{-4} by sending a single, automatically computed, telecommand, thus dramatically reducing the fault isolation time.

1 Introduction

This paper reports on the results of an ESA ITI project, called Ground-basEd diagNostIc sUpport System (GENIUS). The main goal of GENIUS is to demonstrate the advantages of adopting Model-Based Diagnosis (MBD) and active testing for decision support in the operational control of satellites. As a target subsystem, we have used the power generation and distribution network of the Gravity field and steady-state Ocean Circulation Explorer (GOCE) satellite. To the best of our knowledge, GENIUS is the first system to apply a model-based active testing algorithm to a real-world system. The results show that MBD and active testing can increase the diagnostic accuracy and decrease the fault isolation time.

For the diagnosis and the active testing of GOCE, we have used the LYDIA-NG modeling language and the LYDIA-NG suite of algorithms. LYDIA-NG can combine the computations of multiple simulation engines. An example of a simulation engine implemented in LYDIA-NG is the analogue electronic simulator which is based on the well-known Simulation Program with Integrated Circuit Emphasis (SPICE). LYDIA-NG also implements several strategies for the generation of fault candidates and a number of algorithms for active testing. These algorithms are based on AI search and include best-first, and bottom-up greedy search. We discuss all these algorithms and their application to the GOCE satellite.

We validate our approach using nine fault scenarios that summarize a class of failures. These scenarios include single- and multiple-fault injections (up to quadruple faults), masking faults, and faults that cannot be disambiguated by design. We have considered a class of intermittent faults as well.

To independently assess the capabilities of the diagnostic and disambiguation tools, we have created the sensor data for the diagnostic scenarios by implementing a model of the GOCE EPS in the ESA-ESOC SIMSAT infrastructure and not the built-in LYDIA-NG simulation suite. All experiments are performed by using two models of the GOCE Electrical Power System (EPS), one used for the generation of telemetry, and the other used by LYDIA-NG for the tasks of MBD and active testing.

The performance of the diagnostic algorithms has been measured in terms of diagnostic metrics. In our experiments we have no false positive scenarios, no false negative scenarios, the average number of classification errors per scenario is 1.25, and the fault detection time is equal to the computation time. We have further computed an average fault uncertainty of 2.06×10^{-3} which can be automatically reduced to 9.5×10^{-4} by sending a single, automatically computed, telecommand, thus dramatically reducing the fault isolation time.

To facilitate integrating LYDIA-NG with real satellite monitoring systems, we have designed an Application Programming Interface (API) for connecting LYDIA-NG to the GOCE ground telemetry and telecommands infrastructure. This allows the automatic translation of telemetry units and the integration of LYDIA-NG in the satellite ground control system.

The contributions of this paper are as follows:

- We propose a new framework for MBD called LYDIA-NG. LYDIA-NG accommodates algorithms for simulation, diagnosis, and active testing. We measure the diagnostic and disambiguation performance of LYDIA-NG with a number of diagnostic metric. LYDIA-NG provides a way to optimize one or more of these metrics given the specific user environment.
- We build a decision support system on top of LYDIA-

NG. This decision support system is used for diagnosing the EPS of the GOCE satellite.

- We achieve decrease in the fault isolation time of EPS failures by applying active testing.
- We introduce a number of EPS diagnostic scenarios that, in addition to computing performance metrics for our approach, can be also used for comparison to other methods.
- We discuss issues related to the interface of LYDIA-NG with a simulator of telemetry data and a SCADA system for control of satellites.

2 Related Work

LYDIA-NG belongs to a class of MBD methods that use continuous-valued models and sensor data, and use entropy-based methods for test selection to disambiguate diagnoses. It is a generalization of LYDIA, which used discrete-value models.

In terms of diagnostics solvers, LYDIA-NG is related to the HyDE (Hybrid Diagnosis Engine) solver [1]. The HyDE-S variant accepts as input interval-valued hybrid models and continuous-valued sensor data. Another solver, FACT [2], can also use continuous-valued models and sensor data, but requires that the model be represented as a hybrid bond graph. Given an anomaly, FACT first uses an observer-based approach (adopted from the FDI community) with statistical techniques for robust fault detection. Fault isolation is performed using qualitative inference, i.e., by matching qualitative deviations caused by fault transients to those predicted by the model.

There is a large body of work that performs test selection during diagnostics inference. The majority of these approaches require discrete-valued Bayesian models and discrete-valued sensor data. For example, Zheng et al. [3] adopt Bayesian network (BN) models, and interleave diagnostics inference with test selection, which is performed in a greedy manner. Test selection uses loopy belief propagation as the inference method, in order to simultaneously compute approximations of marginal and conditional entropies on multiple subsets of nodes in the BN.

In a similar vein, Wu et al. [4] also use BN methods for test selection, applying their approach to computer networks. Quiao et al. [5] also propose BN test selection methods, but reduce inference complexity by identifying the approximate conditional independence of probes, based on leveraging the conditional independence structure of the BN.

Bellala et al. [6] consider a related framework, except that they allow the presence of noise in the model and the observations. Tolpin and Shimony [7] also frame test selection within a probabilistic model, but propose the use of value of information (VOI) as the criterion for selecting the best test.

In contrast to these approaches, LYDIA-NG does not force the use of a discrete-valued probabilistic model, but can make use of the continuous-valued models widely used in many application domains, such as circuit or energy models.

GENIUS, the design of LYDIA-NG and this paper have been influenced by the International Diagnostic Competition (DXC) [8]. The first years of DXC targeted the Electrical Power System (EPS) testbed in the ADAPT lab at NASA Ames Research Center [9] and we have chosen a real-world satellite subsystem that resembles ADAPT.

In the experiments of this paper we have used a number of performance metrics with which we have measured the performance of our diagnostic and disambiguation algorithms (see Sec. 6.4). These metrics are continuation of the work done for a DXC paper [10] and we propose that future DXC competitions use the up-to-date isolation accuracy and classification error metrics. These metrics assume that the diagnosis is presented as a probabilistic assignment, while in DXC the diagnosis is presented as a set of weighted non-probabilistic assignments. This version simplifies the understanding and the computation of the performance metrics. Further, the uncertainty metric is new. This paper shares with [10] several citations and a small amount of text in the beginning of Sec. 6.4 (introductory part describing the evolution of the metrics). This is done for readability and being self-contained.

3 Concepts and Definitions

In this section we introduce our basic definitions, illustrated on a small artificial electrical circuit that exposes some properties that are present in a real-world power-distribution network of a satellite.

3.1 Running Example

All concepts and algorithms in this paper are illustrated with the help of the circuit shown in Fig. 1. This circuit is best considered as hybrid because there are analogue components such as resistors as well as switches (the latter can, of course, be modeled as non-linear analog electronic components but that would unnecessarily increase the modeling complexity and would not contribute to the diagnostic accuracy).

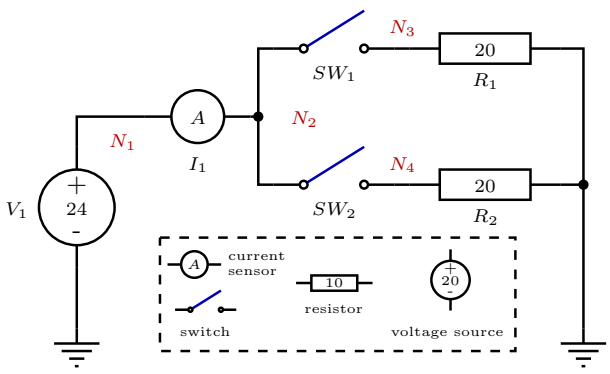


Figure 1: An electrical circuit used as a running example

The circuit in Fig. 1 consists of the voltage source V_1, a current sensor I_1, two normally closed switches SW_1 and SW_2 and the two resistors R_1, and R_2. Notice that if we know the state of the two switches SW_1 and SW_2, the rest of the circuit can be simulated with a circuit simulator like SPICE.

3.2 Models and Diagnoses

We start with the definition of a model. This is a very broad definition (our approach is agnostic to the modeling approach) - we say that a model is a function that computes some set of prediction variables. These prediction variables

are observable from the viewpoint of diagnosis (i.e., they contain sensor data), hence we call them OBS as is traditional in MBD [11]. For our model to be useful for diagnosis and disambiguation, we distinguish two more special subsets of variables: the set of component variables COMPS and the set of command variables CMDS. Both COMPS and CMDS are from Finite Domain Integers (FDI) and their domains are typically small. This limitation is necessary for our AI-based search algorithms to work but limits our approach to relatively abrupt failures and limits our capabilities for graceful degradation and prognostics.

Definition 1 (System Model)**.** A system model M, $\langle$SD, CMDS, COMPS, OBS$\rangle$, is specified using a function SD from two sets of Finite Domain Integer (FDI) variables CMDS and COMPS, to a set of real-valued variables OBS.

In the above definition SD is a function that, when given *full* assignments over CMDS and COMPS, can compute an assignment for the variables in OBS. One can treat CMDS and COMPS as parameters or function selectors. Notice that the function should be specified in such a way so it would result in a valid OBS-assignment for *any* CMDS and COMPS instantiation. In MBD terminology SD is a strong-fault model [12].

The COMPS variables are called component, health, or assumable variables. The CMDS variables are command-variables or user-modifiable inputs (there may be also inputs that the user cannot modify but they have no special meaning for our algorithms). The COMPS-assignment we call a health assignment. The CMDS-assignment we call command-assignment.

We normally associate a *component* with one health variable and, if applicable, a command variable. A resistor, for example, is modeled only with a health variable, while a switch has a command variable that specified the position and a health variable that specifies if the switch is, for example, stuck.

Each health variable has a nominal mode, and the health assignment containing nominal modes only is called the nominal health assignment. In our running example, the nominal health assignment is $\omega^\star = \{V_1 = \text{nominal}, I_1 = \text{nominal}, \text{SW}_1 = \text{nominal}, R_1 = \text{nominal}, \cdots\}$. Note, that even though the health variables are FDI, we usually use symbols instead of numerals for readability. The representation in the implementation of the algorithms, of course, uses small numbers.

Similarly, we have command assignments. There is always a default value for each command variable. For our example in Fig. 1, the switches are normally closed, and the default command assignment is $\gamma = \{\text{SW}_1 = \text{closed}, \text{SW}_2 = \text{closed}\}$.

For the various health (and diagnosis) assignments that we use in this paper we use indexes of the Greek letter ω and for the command assignments we use γ. With all this we can define a hypothetical diagnostic scenario in we can "inject" a hypothetical health (fault) assignment $\omega^\star$.

Definition 2 (Diagnostic Scenario)**.** A diagnostic scenario SCN, $\langle M, \gamma, t^\star, \omega^\star, \alpha\rangle$, is specified using a system model M, an assignment $\omega^\star$ to the health variables in COMPS at time $t^\star$, and an assignment γ to the user-command variables in CMDS.

What is new in the above definition is the use of the observation assignment α. We assume that α is a real vector but keep the notation compatible with other AI papers where the observation is a Boolean assignment [13].

In what follows we define diagnosis.

Definition 3 (Diagnosis)**.** A diagnosis DIAG, $\langle M, \omega, t\rangle$ is defined as a probabilistic assignment ω to the variables in COMPS at time t.

A diagnosis is very similar to a health assignment, except that there is a probability for every literal. Let us say, that in our running example, the two of the resistors are open-circuited with the same probability of 0.25. The expression for ω looks like this: $\omega = \{\Pr(R_1 = \infty) = 0.25, \Pr(R_2 = \infty) = 0.25\}$.

In the definition of diagnostic scenario and diagnosis we also have the time of the fault injection $t^\star$ and the time of the diagnosis t. These are rarely used in our exposition as we construct most of the diagnostic scenarios in such a way as to have one fault injection and assume one diagnosis, which, if true (positive or negative), would have $t > t^\star$. We will need t and $t^\star$, though, when we define and compute the false-positive and false-negative scenario metrics.

Using the three definitions from above, we can essentially "run" diagnosis and disambiguation (we will shortly make the notion of disambiguation more precise). Clearly, the goal of a diagnostic system would be, given a diagnostic scenario, to compute a diagnosis ω that is close to $\omega^\star$. Of course, the diagnostic system cannot use the fault injection $\omega^\star$. The latter fault injection is only used while "training" and evaluating the whole diagnostic system. Finally, we need a function that will compute the "distance" between the fault injection $\omega^\star$ and the diagnosis ω. Such functions are well-studied in diagnosis, reliability, and others, and are called "metrics".

Definition 4 (Diagnostic Metric)**.** A diagnostic metric DM, $\langle F$, SCN, DIAG$\rangle$ is defined using a function F that maps $(t^\star, \omega^\star, t, \omega)$ into $[0, \infty)$.

The computation of metrics allows users to compare the performance of various diagnostic and simulation approaches, etc. The whole task of diagnosis and disambiguation can be cast as an optimization problem that aims at minimizing one or more metrics.

4 Diagnosis

In what follows we show an algorithmic framework for computing diagnoses from models and scenarios. This is the first step in a two-step process in which (1) a diagnostics algorithm computes a diagnosis given a model and an observation and (2) if the diagnosis is ambiguous, i.e., one or more components are said to be in a certain fault state with probability different from one, then a disambiguation algorithm is used to decrease the diagnostic uncertainty. This section concerns the first step (diagnosis) while the next section describes the disambiguation process which involves the computation and application of new commands.

4.1 Overview

The basic idea of the LYDIA-NG diagnostic library (shown in Fig. 2) is to perform multiple simulations for various hypothesized health states of the plant. The output of these multiple simulations is then processed and combined into single diagnostic output.

The LYDIA-NG diagnostic library consists of the following building blocks:

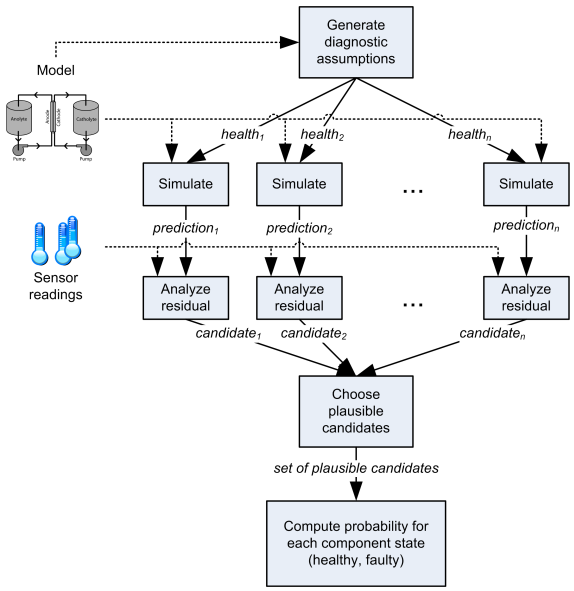


Figure 2: Overview of the LYDIA-NG diagnostic method

Generator of Diagnostic Assumptions: A diagnostic assumption is a set of hypothetical assignments for the health or fault state of each component in the system. The "all nominal" diagnostic assumption assigns healthy status to each component. LYDIA-NG allows one nominal and one or more faulty states per component.

Simulation Engine: Given a diagnostic assumption, LYDIA-NG can construct a simulation model of the system. This simulation model consists of equations. By solving this system of equations LYDIA-NG computes values for one or more *observable* variables. The values of these observable variables is also referred to as a *prediction*.

Residual Analysis Engine: A prediction is compared to the sensor data by a residual analysis engine. This engine combines the individual discrepancies in each sensor data/predicted variable pair to produce a single real value that indicates how close is the prediction of the simulation engine to the sensor data obtained from the plant. A simulation that results in all predicted values coincide with the measured ones will result in the residual being zero. The data structure containing predictions, their corresponding sensor data and the computed residual is called a *diagnostic candidate* or simply *candidate*.

Candidate Selection Algorithm: Not all candidates generated by the residual analysis engine are used for computing the final system health. The candidate selection algorithm discards each candidate whose residual is larger than the residual of the "all nominal" candidate.

System State Estimation Algorithm: LYDIA-NG uses the set of candidates that is computed by the candidate selection algorithm to compute an estimate for the health of each component. This is done by the system state estimation algorithm. Finally, LYDIA-NG computes RCoF by choosing the components with highest probability of failure.

4.2 Algorithm

Algorithm 1 shows the top-level diagnostic process. The inputs to Alg. 1 are a model and a scenario, and the result is a diagnosis.

At the hear of Alg. 1 is the use of simulation. Algorithm 1 supports a large variety of simulation methods that may or may not use time as an independent variable. In the setup described in this paper we have used SPICE in combination with a constraint propagation solver. The latter we have used for sensor values, complex components such as mixed analog-digital electronics and other parts of the model where it is difficult or inappropriate to model with SPICE. The only requirement toward the simulation engine is to predict a number of variables whose types can be mapped to LYDIA-NG and to be relatively fast (the computational performance of LYDIA-NG will not be thoroughly discussed in this paper which emphasizes the application of LYDIA-NG to a space model).

Algorithm 1 Diagnosis framework

```
1: function DIAGNOSE(SCN) returns a diagnosis
       inputs: SCN, diagnostic scenario
       local variables: h, FDI vector, health assignment
                        p, real vector, prediction
                        Ω, a set of diagnostic candidates
                        DIAG, diagnosis, result
2:     while h ← NEXTHEALTHASSIGNMENT() do
3:         p ← SIMULATE(M, γ, h)
4:         r ← COMPUTERESIDUAL(p, α)
5:         Ω ← Ω ∪ ⟨h, r⟩
6:     end while
7:     DIAG ← COMBINECANDIDATES(Ω)
8:     return DIAG
9: end function
```

The basic idea of Alg. 1 is to simulate for various health assignments and to compare the predictions with the observed sensor data (i.e., telemetry). There are several important aspects of this algorithms that ultimately affect the diagnostic accuracy as measured by various performance metrics (see Sec. 6.4).

The first algorithmic property that determines many of the diagnostic performances is the order in which health-assignments are generated. In Alg. 1 this is implemented in the NEXTHEALTHASSIGNMENT function. The latter subroutine also determines when to stop the search and should be properly parametrized depending on the model and the user requirements. In the standard LYDIA-NG diagnostic library we provide the following diagnostic search policies:

Breadth-First Search (BFS): This policy first generates the nominal health assignment, then single-faults, double-faults, etc.

Depth-First Search (DFS): This search policy starts with the nominal health assignment, then adds a single-fault, continues with a double fault including the first, and so on, until all components are failed. After the all-faulty assignment is generated, the algorithm backtracks one step and generates a sibling assignment and continues traversing down and backtracking in the same manner until no more backtracking is possible.

Backwards Greedy Stochastic Search (BGSS): In this mode, the search start from the all-faulty assignment.

A random health variable is then flipped and the flip is retained iff the flip leads to a decrease in the residual. The order of health variables is arbitrary. As the whole search process is stochastic, it needs to be run multiple iterations in order to achieve the desired completeness. A formal description of this method for Boolean circuit models can be found in [13].

Each simulation produces what we call a *candidate*: a set of predicted values for a given health-assignment. The second important property of Alg. 1 is the comparison and ordering of the diagnostic candidates. This is done by mapping the predicted and observed variables into a single real-number, called *residual*. The residual computation is discussed in what follows.

4.3 Residual Generation

Residual generation functions in LYDIA-NG bear resemblance to loss functions in decision theory.

Definition 5 (Residual Function)**.** A residual function R maps a prediction vector $\mathbf{p}$ and an observation vector α into a real-number $[0; \infty)$.

We next show two straightforward residual generation functions.

Squared Residuals:

$$R_{\text{sq}}(\text{OBS}, \mathbf{p}, \alpha) = \sum_{v \in \text{OBS}} W(v) \left[\mathbf{p}(v) - \alpha(v)\right]^2 \quad (1)$$

where $W(v)$ is a weight-value associated with sensor v, $\mathbf{p}(v)$ is the value of variable v in the prediction assignment $\mathbf{p}$ and $\alpha(v)$ is the value of the observable variable v.

Absolute Residuals:

$$R_{\text{abs}}(\text{OBS}, \mathbf{p}, \alpha) = \sum_{v \in \text{OBS}} W(v) \left|\mathbf{p}(v) - \alpha(v)\right| \quad (2)$$

where $W(v)$, $\mathbf{p}(v)$ and $\alpha(v)$ are used in the same way as in Eq. 1.

A disadvantage of the squared residuals function R_{sq} is that it adds a lot weight to outliers. In decision theory, the absolute loss function that corresponds to the R_{abs} function is discontinuous. The latter, however, is not a problem for the algorithms described in this paper and we prefer R_{abs} over R_{sq}.

4.4 Computation of Component Failure Probabilities

Consider the circuit shown in Fig. 1 and a scenario $\alpha = \{I_1 = 1.19\}$. This scenario corresponds to one of the resistors being open-circuited or one of the switches being stuck-open. Table 1 shows applying Eq. 2 for the predictions simulated from the nominal and all single-fault health assignments. The rows of Table 1 are sorted in order of an increasing residual value. In this table (and below) we abbreviate a stuck switch as S and an open-circuit resistor mode as OC.

The COMBINECANDIDATES subroutine from Alg. 1 uses a table similar to the one shown in Table 1. It retains only the predictions with residuals smaller than the residual of the nominal prediction. The reason for that is that the nominal prediction is the only one that has a special meaning in LYDIA-NG and leads to a "landmark" residual, i.e., LYDIA-NG does not attempt to differentiate amongst the various

Table 1: Single-fault residuals for the circuit shown in Fig. 1 and an observation simulated from a single open-circuited resistor

V_1	I_1	SW_1	SW_2	R_1	R_2	faults	R_{abs}
–	–	S	–	–	–	1	0.0006
–	–	–	S	–	–	1	0.0006
–	–	–	–	OC	–	1	0.0006
–	–	–	–	–	OC	1	0.0006
–	–	–	–	–	–	0	1.1758
F	–	–	–	–	–	1	1.1888
–	F	–	–	–	–	1	1.1888
–	–	–	–	SC	–	1	79.3402
–	–	–	–	–	SC	1	79.3402

fault-mode predictions. As a result, in our running example, only the first four rows of Table 1 are considered when calculating the final fault-probabilities.

The second step of COMBINECANDIDATES is to convert R_{abs} in the interval $[0; 1]$ where $R_{\text{norm}} = 0$ for the nominal prediction and $R_{\text{norm}} = 1$ for a fault prediction that gives $R_{\text{abs}} = 0$. Applying this on Table 1 gives us Table 2.

Table 2: Normalized single-fault residuals from Table 1 that are smaller than the nominal residual

SW_1	SW_2	R_1	R_2	R_{norm}
S	–	–	–	1
–	S	–	–	1
–	–	OC	–	1
–	–	–	OC	1

Finally, what remains to be done is to normalize the rightmost column of Table 2 so it sums up to one and marginalize the probability of failure in each column. For the small circuit we are analyzing this results in $\{\Pr(\text{SW}_1 = \text{S}) = 0.25, \Pr(\text{SW}_2 = \text{S}) = 0.25, \Pr(R_1 = \text{OC}) = 0.25, \Pr(R_2 = \text{OC}) = 0.25\}$. The fact that all probabilities are 0.25 means that Alg. 1 cannot determine unambiguously which component is the faulty one. In this case this is due to the fact that there is only one sensor, i.e., the unambiguity is due to sensor placement and circuit design.

One way to reduce this ambiguity is to change the position of SW_1 and/or SW_2. In the next section we devise an algorithmic framework that works for *any* circuit or model that can be diagnosed in the LYDIA-NG framework.

5 Disambiguation

An ambiguous scenario is when LYDIA-NG cannot be certain if a component is failing or not (we will shortly discuss a more precise notion of uncertainty). The reasons for that can be in the design of the system itself, due to model approximation, or due to the choice of the diagnostic algorithm. Consider, for example, the simple electrical circuit shown in Fig. 1. Due to the fact that there is a single current sensor in the design (I_1), an open-circuit resistor R_1 cannot be distinguished from an open-circuit R_2, SW_1, or SW_2. A set of modes that cannot be distinguished from each other is called an *ambiguity group*. Sometimes ambiguous results

are due to the observation (consider the case in which there is no sensor data at all). In other cases, ambiguity is a result of the artifact design as shown in Fig. 1. In these cases no diagnostic algorithm can return a unique diagnosis. Finally, the *diagnosability* can be influenced by modeling approximation. Consider an alternative of the circuit shown in Fig. 1, in which R_1 is 22 Ω but it is modeled as a 20 Ω resistor.

Ambiguity groups appear only for certain plant configurations (positions of switches SW_1 and SW_2 in our example). Let us consider the case in which both SW_1 and SW_2 in Fig. 1 are closed and the only malfunctioning component is R_1, where R_1 is open-circuited. As a result of the fault, the current sensor I_1 shows -1.2 A[1] instead of the nominal -2.4 A. The most informed diagnosis in this case is that both R_1 and R_2 are equally-likely to be open-circuited. An equivalent statement is that given the single measurement of $\mathrm{I}_1 = -1.2$ A, both R_1 and R_2 fail with probability of 0.5 (to keep the example short we do not allow SW_1 and SW_2 to fail, LYDIA, however, allows switches to fail).

A LYDIA-NG diagnosis contains a discrete probability for each component mode in the system. Each component typically specifies a single nominal mode and one or more fault modes. Consider, for example, R_1 that can be either in nominal mode ($R_1 = 20$), or in short-circuited mode ($R_1 = 0$), or in open-circuited mode ($R_1 = \infty$). We will use the following notation for the health of R_1:

$$\begin{array}{l}\Pr(R_1 = 20) = 0.5,\\ \Pr(R_1 = 0) = 0,\\ \Pr(R_1 = \infty) = 0.5\end{array} \tag{3}$$

For brevity, we omit the zero-probability assignments:

$$\Pr(R_1 = 20) = 0.5, \Pr(R_1 = \infty) = 0.5. \tag{4}$$

In (4), the diagnostic engine does not really know if R_1 is healthy or open-circuited. A much more preferred situation from the diagnostic viewpoint would be:

$$\Pr(R_1 = \infty) = 1 \tag{5}$$

In (5) the diagnostic engine has determined a unique mode (R_1 is open-circuited), and there is no *uncertainty* in the diagnosis. We can derive a formula that gives a quantitative *metric* for the uncertainty in the health assignment of a component C:

$$U(C) = \sum_{x \in C^\star} -\Pr(C = x) \lg_{|C^\star|} \Pr(C = x), \tag{6}$$

where $\Pr(C = x)$ denotes the probability of a component C being in a healthy/faulty state x and $C^\star$ is the set of all possible component states of C.

We can compute the uncertainty of a diagnostic assignment for the whole system as the average uncertainty of all of its components:

$$\bar{U} = \frac{1}{|\mathrm{COMPS}|} \sum_{C \in \mathrm{COMPS}} U(C), \tag{7}$$

where COMPS is the set of all components in the system.

If we treat each component C as a random variable (the sum of the probabilities of all component modes is one), then Eq. (6) represents the average component health *entropy* and Eq. (7) represents the average health entropy of the whole system. The values $U(C)$ and $\bar{U}$ are always in the interval $[0; 1]$ where 0 is the lowest uncertainty, and 1 is the highest, i.e., the probability for each component state is the same if and only if the entropy equals one.

Given an uncertain diagnosis, the LYDIA-NG disambiguation library computes new *plant-configuration assignment* (user-modifiable inputs) that optimally reduces the average uncertainty in the next diagnosis. The *next* (future) diagnosis, however, depends on a future observation which is unknown. To estimate this future observation, LYDIA-NG uses the existing (current) diagnosis.

Recall that a diagnosis is a probability assignment over the component states in the system. For a simulation, however, we need a deterministic assignment over the set of assumable (health) variables. For the simulation of a diagnosis, LYDIA-NG averages the observations obtained by failing all components that appear in the diagnosis with nonzero probability. This is a weighted average and the weight for each simulation is the probability of failure as it appears in the diagnosis. By doing this, LYDIA-NG uses all information in a diagnosis for estimating a future observation.

Consider the circuit in Fig. 1 and a diagnosis

$$\Pr(R_1 = \infty) = 0.5, \Pr(R_2 = \infty) = 0.5 \tag{8}$$

The LYDIA-NG disambiguation algorithm first simulates the circuit with R_1 open-circuited, R_2 nominal, SW_1 closed, and SW_2 open. The predicted value for I_1 is 0 A. A simulation with R_1 nominal, R_2 open-circuited, SW_1 open, and SW_2 closed, results in the same predicted future observation of 0 A for I_1. As both simulated failures appear in (8) with the same probability the next predicted observation is $\mathrm{I}_1 = 0$ A. What remains for LYDIA-NG is to apply (6) to the diagnoses computed with the two observations just described. The first diagnosis is:

$$\Pr(R_1 = \infty) = 1 \tag{9}$$

while the second diagnosis is:

$$\Pr(R_2 = \infty) = 1 \tag{10}$$

Plugging (9) or (10) in (7) would maximize $\bar{U}$ which is intuitively correct as opening either SW_1 or SW_2 would result in current flowing through one resistor only, and hence disambiguate the resulting diagnosis. LYDIA-NG presents this kind of reasoning to the user who is advised to open SW_1 or SW_2 if she wants to obtain more diagnostic information than the one contained in (8).

A concluding remark is that by changing the positions of SW_1 and SW_2 we obtain a *virtual* (current) *sensor* that can be used for more precise determination of the health state of resistors R_1 and R_2.

LYDIA-NG can disambiguate in the case of *non-intermittent* failures only. Disambiguation of intermittent behavior is significantly more complicated and subject of future research. A subject of future development is also the automated computation of worst-case diagnosability of a model.

Algorithm 2 shows the LYDIA-NG disambiguation framework. It uses Alg. 1 as a diagnostic oracle.

The main loop of Alg. 2 (1) simulates the effects of a number of command assignments, (2) computes diagnosis and (3) takes the command assignment that results in a minimal diagnostic entropy.

[1]The current is negative so this document matches the LYDIA-NG implementation of the example and the passive sign convention [14].

Algorithm 2 Disambiguation framework

```
1: function DISAMBIGUATE(SCN) returns a command
       inputs: SCN, diagnostic scenario
       local variables: DIAG, diagnosis
                        d, component health variable
                        h, FDI vector, health assignment
                        α, real vector, observation
                        γ', γ_min, command assignments
                        H, H_min, reals, entropy
2:     DIAG ← DIAGNOSE(SCN)
3:     while γ' ← NEXTCOMMAND(γ) do
4:         H ← 0
5:         for all {d ∈ DIAG : Pr(d) > 0} do
6:             h ← MAKEHEALTHASSIGNMENT(d)
7:             α ← SIMULATE(DM, γ, h)
8:             DIAG' ← DIAGNOSE(⟨M, γ', t, ω*, α⟩)
9:             H ← H + ENTROPY(DIAG')Pr(d)
10:        end for
11:        if H < H_min then
12:            H_min ← H
13:            γ_min ← γ'
14:        end if
15:    end while
16:    return γ_min
17: end function
```

When describing Alg. 2 we use *current* (diagnosis, observation, etc.) to denote the state before a new user command has been applied and *predicted* to denote observation, diagnoses, entropies, etc., after the new user command has been executed.

Algorithm 2 starts by computing the current diagnosis (line 2). The subroutine NEXTCOMMAND iterates over the space of all possible user commands. Notice that it takes as an argument the current state of all command variables and returns a new vector γ' that reflects the chosen command. If, for example, $\gamma = \{\mathrm{SW}_1 = \mathrm{closed}, \mathrm{SW}_2 = \mathrm{closed}\}$, executing the user command "open switch one", results in $\gamma' = \{\mathrm{SW}_1 = \mathrm{open}, \mathrm{SW}_2 = \mathrm{closed}\}$.

Depending on how NEXTCOMMAND is implemented, the following command generation strategies have practical significance:

Breadth-First Search (BFS): First, all possible changes to one command variable are considered, then all possible pairs, triples, etc. of command variables are generated. Clearly, this strategy quickly causes a combinational blow-up, hence for larger systems, it is feasible to consider only changes to single command variables.

Greedy Stochastic Search (GSS): This policy starts with a single change to a command variable. If this change leads to an entropy reduction (compared to the entropy of the current diagnosis), then this changed is preserved and a second command variable is changed. The process is repeated until changes to single command variables lead to a reduction in the predicted entropy. The order in which changes are applied is random, hence, the algorithm is stochastic. The disadvantage of this method is that it is incomplete, i.e., certain *combinations* of changes to command variables will not be considered given limited computational resources.

There are more command generation policies such as a fully stochastic generation of commands. This is hardly of practical interest as usually there is some cumulative cost related to combining individual changes to command variables into a multi-variable command and fully stochastic selection of changes would hardly result in minimal entropy.

One can also implement Backward Greedy Stochastic Search (BGSS). In this command generation strategy, the algorithm starts with changing all possible command variables in γ' and greedily flips the individual variables to their original values in γ. Again, a command variable flip should be accepted only if it reduces the predicted entropy. This strategy is suitable only for a class of devices that are configurable through multi-variable command changes.

In line 5 of Alg. 2 we iterate over each failing component d in the current diagnosis as computed by Alg. 1 in line 2 (each component that fails with non-zero probability). MAKEHEALTHASSIGNMENT (line 6) is an auxiliary function that returns a health assignment that has d failing with its respective failure mode and all other component variables nominal. The current command assignment γ and the health assignment $\mathbf{h}$ are supplied to the simulator that is invoked in line 7. This is the same simulation engine that is used in Alg. 1. Finally, the ENTROPY function in line 9 is the implementation of Eq. 7.

The conditional in lines 11 - 14 select the user command γ' that minimizes the expected entropy H. This is collected int the $\gamma_{\min}$ variable which is returned in line 16. In the real world, the first k-lowest entropy γ' commands are presented to the user so she can choose the one that best matches the situation.

6 Experiments

In this section we describe experiments with the LYDIA-NG framework in the scope of the GENIUS project. To make our study specific we have modeled the EPS of GOCE but the results are valid to a wider-class of satellite power-supply systems.

6.1 Exprimental Test-Bed

The experimental test-bed for LYDIA-NG is shown in Fig. 3. We have used the ESA-ESOC SIMSAT simulator framework to generate the failure scenarios for LYDIA-NG. GOCE uses the SCOS-2000 SCADA system and the LYDIA-NG diagnostic library interfaces it. The SIMSAT can be connected to SCOS-2000 via the Ground Models (a SIMSAT library that provides telemetry packet marshalling).

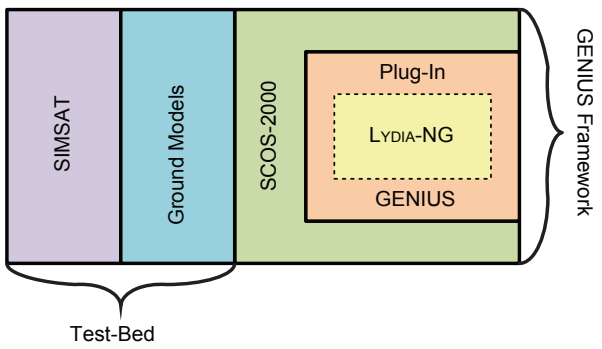


Figure 3: Overview of the GENIUS test-bed

6.2 GOCE EPS Model

From the viewpoint of modeling, the GOCE EPS has three major subsystems:

Solar Array Panels and Electronics: There are six body-mounted solar array panels. The panels are connected to the solar panel electronics and their voltage and current sensors are reported in the telemetry. Each solar array panel is connected to a Power Conditioning Unit (PCU). Each PCU contains, in addition to input and output voltage and current sensors, a Maximum Power Point Tracker (MPPT), and a solar array regulator. The PCU electronics is double and triple redundant. We have not considered scenarios leading to ambiguous diagnoses with faulty components in the PCU units.

Battery and Battery Control: The battery is connected to the main bus through a pair of switches that are not used in normal operation. There are redundant charging and discharging voltage and current sensors. The battery charge control electronics has the task of keeping the battery charged while extending its life.

Power Distribution Network and Thermal Control: There are two times eight heater groups (nominal and redundant), each group consisting of six heaters. Each heater can be individually controlled via a Transistor SWitch (TSW). The positions of the TSWs are reported in the telemetry. The heater groups are connected to the main bus through Latch Current Limiters (LCLs). There is a current sensor per heater group.

The LYDIA-NG model is ≈ 1000 lines and the number of variables is typically large for an MBD application (see Table 3).

Table 3: GOCE EPS model properties

Model Property	Value
observable variables	186
health variables	289
command variables	132
internal variables	790
total number of variables	1397

6.3 Diagnostic Scenarios

Table 4 shows a summary of the GENIUS diagnostic scenarios with which we have tested LYDIA-NG.

We have arbitrarily chosen heater group A8 for experimentation—the results do not change if we choose another group or a combination of groups.

TST1: There is no fault injection during this scenario. This scenario is used to check the diagnostic algorithms for spurious diagnoses (false positive scenarios).

TST2: In this scenario a single PCU current sensor is failed. The fault-mode is out-of-bound value. Such a fault can be identified unambiguously by a diagnostic algorithm.

TST3: This scenario injects a single open-circuited heater in the first heater group. Due to the fact that there is only a single current-sensor per heater group and that there are heaters of the same type (resistance), this fault cannot be identified unambiguously by any diagnostic algorithm.

Table 4: GOCE EPS fault scenarios

Name	Class	Faults
TST1	nominal	–
TST2	single-fault	current sensor
TST3	single-fault, ambiguous	heater
TST4	continuation of TST3	heater
TST5	multiple-faults	current sensors
TST6	multiple-faults, related	current sensor and a switch
TST7	multiple-faults, ambiguous	solar arrays and heaters
TST8	multiple-faults, degradation	solar arrays
TST9	multiple-faults, accumulation	solar array and a voltage sensor
TST10	slow degradation	heater
TST11	intemittent	switch

TST4: This is a continuation of TST3. After the first injection, the user is assumed to send a telecommand that switches-off one heater. This user-command now becomes part of the scenario. As a result a diagnostic algorithm should be able to produce a less-ambiguous diagnosis after the user-command.

TST5: In this scenario we inject simultaneously two current sensor faults of the same type as in TST2.

TST6: In this scenario we inject a double-fault. The two faults are a current sensor in the heater group and a switch. The two faults are topologically close, i.e., they are in the same heater group.

TST7: This is a quadruple-fault where the faults are relatively far from each other in the device topology. The scenario is highly-unlikely and is used for "stress-testing" the performance of the diagnostic and disambiguation algorithms. In this scenario we have two of the solar arrays suffer from the loss of multiple solar cells and two open-circuited heaters.

TST8: This is a double fault degradation, i.e., two of the solar-panels expose loss of power.

TST9: This scenario starts with degrading the capacity of a single solar-panel and then adds a failed voltage sensor. The individual faults accumulate and they are close topologically (i.e., one may have led to the other).

TST10: This scenario is not implemented at the time of the writing of this paper. It is included for planned future work on graceful-degradation, prognostics and other areas related to MBD. The idea is to have a heater that changes its resistance due to aging. At this moment the authors have not studied the exact physics of failure of such faults.

TST11: This is a scenario in which faults appear and disappear in consecutive measurements. The injected fault is a faulty LCL switch.

6.4 Metrics of Diagnostic Performance

The metrics for evaluating Diagnostic Algorithm (DA) performance depend on the particular use of the diagnostic system, the users involved, and their objectives.

Several institutions and organizations have proposed metrics that measure diagnostic performance [15; 16; 17; 18; 19; 20; 21]. Among those, the SAE's "Health and Usage Monitoring Metrics" [15] defines probability of detection and probability of false alarms as key indices for evaluating diagnostic algorithm performance.

In [19], the performance metrics are defined separately for detection, and isolation. For detection, the metrics include thresholds, accuracy, reliability, sensitivity to load, speed, or noise, and stability. The isolation metrics include the detection metrics, but also include measures for discrimination and repeatability.

In LYDIA-NG, we make a distinction between detection, isolation, and computational performance and highlight metrics for each category. In general several other classes of metrics are possible, including cost/utility metrics, effort metrics (in building systems for example) and also other categories such as fault identification and fault recovery metrics. The expectation is that as LYDIA-NG evolves a comprehensive list of desired metric classes and categories will be developed to aid framework users in choosing the performance criteria they want to measure.

Table 5: Metrics summary

Metric	Name	Class
M_{fd}	fault detection time	detection
M_{fn}	false negative scenario	detection
M_{fp}	false positive scenario	detection
M_{da}	scenario detection accuracy	detection
M_{fi}	fault isolation time	isolation
M_{err}	classification errors	isolation
M_{ia}	isolation accuracy	isolation
M_{ent}	diagnostic uncertainty	isolation
M_{cpu}	CPU load	computational
M_{mem}	memory load	computational

The ten metrics we have defined are summarized in Table 5. All metrics are real numbers (M_{mem} is an exception as it measures a discrete value - the number of bytes that are used). Several metrics (M_{fn}, M_{fp}, M_{da}, M_{err}, M_{ia}, and M_{utl}) are defined in such a way as to produce a number in the interval $[0; 1]$. We show that isolation accuracy is dual to classification errors and, as M_{ia} does not contain any additional information to M_{err}, LYDIA-NG does not compute it.

Detection Metrics

The distinction between detection and isolation has practical importance. A DA may announce a fault detection before it knows the root cause of failure (for example, a detection announcement can be based solely on surpassing sensor threshold values). A detection signal cannot be retracted by a DA while it is legal to retract an isolation announcement when more faults are expected (at the of writing of this LYDIA-NG does not support retractions but such feature may be added in the future). The detection metrics include:

Fault Detection Time: The *fault detection time* (the reaction time for a diagnostic engine to detect an anomaly) is directly measured as:

$$M_{\mathrm{fd}} = t - t^{\star} \tag{11}$$

The fault detection time is reported in milliseconds and is computed only for non-nominal scenarios for which a DA computes a diagnosis at least once.

False Negative Scenario: The *false negative scenario* metric measures whether a fault is missed by a diagnostic algorithm and is defined as:

$$M_{\mathrm{fn}} = \begin{cases} 1, & \text{if } t = \infty \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

M_{fn} of a nominal scenario is defined to be zero.

False Positive Scenario: The *false positive scenario* metric penalizes DAs which announce spurious diagnoses and is defined as:

$$M_{\mathrm{fp}} = \begin{cases} 1, & \text{if } t < t^{\star} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where $t^{\star} = \infty$ for nominal scenarios (i.e., scenarios during which no fault is injected).

Note that the above two metrics (M_{fn} and M_{fp}) are computed for each scenario and their computation is based on the times of injecting and announcing the fault. We also have false negative and false positive components in the context of individual diagnostic candidates (recall that a DA sends a set of diagnostic candidates at isolation time) which we will discuss later in this document.

Scenario Detection Accuracy: The *scenario detection accuracy* metric is computed from M_{fn} and M_{fp}:

$$M_{\mathrm{da}} = 1 - \max(M_{\mathrm{fn}}, M_{\mathrm{fp}}) \tag{14}$$

M_{da} is 1 if the scenario is true positive or true negative and 0 otherwise (equivalently, $M_{\mathrm{da}} = 0$ if $M_{\mathrm{fn}} = 1$ or $M_{\mathrm{fp}} = 1$, and $M_{\mathrm{da}} = 1$ otherwise). M_{da} splits all scenarios into "true" and "false". Incorrect scenarios are further classified into false positive (M_{fp}) and false negative (M_{fn}). Correct scenarios are true positive if there are injected faults and true negative otherwise (the latter separation into true positives and true negatives is rarely of practical importance).

Isolation Metrics

Consider a fault injection ω. We construct the *set* of failing components $\bar{\omega}$ by taking the identifiers of the failing components only. In other words, we discard the precise fault state of a failing component, and only retain the information that this component is in an off-nominal state.

We can do something similar from a diagnosis $\omega^{\star}$. Of course, in this case we will take only components that are *failing* with non-zero probability. We denote the set of failing components in a diagnosis ω as $\bar{\omega}$.

Notation-wise, we put a bar above a fault injection $\omega^{\star}$ or a diagnosis ω to take two sets of failing components - the fault $\bar{\omega}^{\star}$ and the candidate $\bar{\omega}$. We can now use those two to explain false positives and false negatives in the context of multiple-fault isolation accuracy metrics.

Both the candidate $\bar{\omega}$ and the injected fault $\bar{\omega}^{\star}$ are sets of components. The intersection of those two sets are the

properly diagnosed components. The false positives are the components that have been considered faulty but are not actually faulty. The false negatives are the components that have been considered healthy but are actually faulty. Figure 4 shows how $\bar{\omega}$ and $\bar{\omega}^\star$ partition all components into four sets.

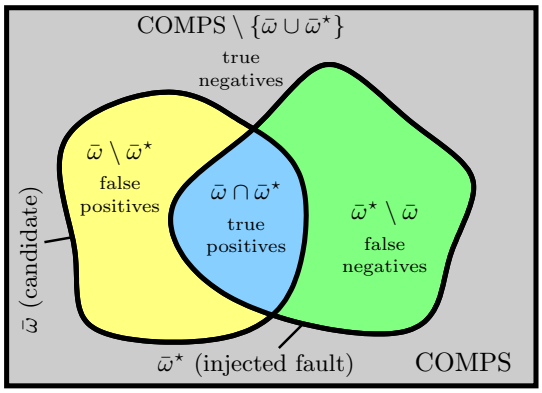


Figure 4: The diagnostic candidate $\bar{\omega}$ and the injected fault $\bar{\omega}^\star$ partition COMPS into four sets

False positives and false negatives in this context relate to individual candidates, i.e., misclassified components in a single diagnostic candidate. There are also scenario-based false negative and false positive metrics (defined earlier in this section), which summarize whole scenarios and are not to be confused with the false positives and false negatives in the context of isolation metrics.

For brevity, in the remaining of this document we use the notation in Table 6 for the Fig. 4 sets.

Table 6: Notation for sizes of some frequently used sets

Var.	Set	Description
f	$\lvert\text{COMPS}\rvert$	all components
n	$\lvert\bar{\omega}^\star \setminus \bar{\omega}\rvert$	false negatives
N	$\lvert\text{COMPS} \setminus \bar{\omega}\rvert$	the set of healthy components from the viewpoint of the DA
$\bar{n}$	$\lvert\bar{\omega} \setminus \bar{\omega}^\star\rvert$	false positives
$\bar{N}$	$\lvert\bar{\omega}\rvert$	the set of faulty components from the viewpoint of the DA

Based on the representation given in Figure 4, the meaning of false positives and false negatives can be interpreted differently depending on what the diagnosis results are supporting (abort decisions, ground support, fault-adaptive control, etc.). Researchers have proposed different methods to assess the meaning of isolation accuracy and its practical and economical implications.

[16] introduced metrics based on the receiving operating characteristic (ROC) analysis [18], which illustrates the trade-off space between the probability of false alarm and the probability of detection for different signal to noise ratio (SNR) levels. The method is used to test the relative accuracy of diagnostic systems based on different threshold settings. Later, they also proposed a combined metric [17] that accounts for consequential event costs including missed detection, false alarms, and misdiagnosis. Another widely used metric for isolation accuracy is the Kappa Coefficient [15]. It is based on the construction of a confusion matrix that summarizes diagnostic results produced by a reasoner over a number of test/use cases. In essence, the Kappa Coefficient measures the ability of an algorithm to discriminate among many fault candidates.

In this document, we take a simplistic approach and assume that false positives and false negatives have an equal cost for the diagnostic task and operations. The isolation metrics include:

We can now return to using the original $\omega^\star$ and ω, i.e., we don't need their set equivalents $\bar{\omega}^\star$ and $\bar{\omega}$.

Classification Error: Given a fault injection $\omega^\star$ and a diagnosis ω, the *classification error* metric is defined as:

$$M_{\text{err}} = \sum_{C \in \text{COMPS}} |I(C = x) - \Pr(C = x)| \tag{15}$$

where

$$I(C = x) = \begin{cases} 1, & \text{if } (C = x) \in \omega \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

In Eq. (15), $|I(C = x) - \Pr(C = x)|$ denotes the symmetric difference of the ω and $\omega^\star$ assignments and can be interpreted as the probabilistic equivalent of the number of misclassified components.

We can define a new metric, the isolation accuracy M_{ia}:

$$M_{\text{ia}} = \sum_{C \in \text{COMPS}} |1 - I(C = x) - \Pr(C \neq x)| \tag{17}$$

One can easily see that M_{ia} and M_{err} are duals, i.e.:

$$\frac{M_{\text{ia}}}{f} + \frac{M_{\text{err}}}{f} = 1 \tag{18}$$

The isolation accuracy metric M_{ia} originates in the automotive industry [15]. The Aerospace Recommended Practice (ARP) computes the closely related probability of correct classification in the following way. For each component we compute the square confusion matrix. The probability of correct classification is the sum of the main diagonal divided by the total number of classifications (see the referenced ARP [15] for details and examples).

Diagnostic Uncertainty: Unlike M_{err} and M_{ia}, the diagnostic uncertainty metric M_{ent} is computed from the diagnosis ω only. As a result, the diagnostic uncertainty is not suitable for being used on its own, but should be used in combination with M_{ia} and M_{err}.

We start by defining the diagnostic uncertainty of a component $C \in \text{COMPS}$:

$$U(C) = \sum_{x \in C^\star} -\Pr(C = x) \lg_{|C^\star|} \Pr(C = x), \tag{19}$$

where $\Pr(C = x)$ denotes the probability of a component C being in a healthy/faulty state x and $C^\star$ is the set of all possible component states of C.

We can compute the uncertainty of a diagnostic assignment for the whole system as the average uncertainty of all of its components:

$$M_{\text{ent}} = \frac{1}{f} \sum_{C \in \text{COMPS}} U(C), \tag{20}$$

where COMPS is the set of all components in the system.

Fault Isolation Time: Consider a fault injection $\omega^\star$ and a sequence of diagnoses $\omega_1, \omega_2, \ldots, \omega_n$. The fault isolation time M_{fi} is defined as:

$$M_{\text{fi}} = t' - t^\star \tag{21}$$

where t' equals the first time t_i ($i = 1, 2, \ldots, n$) in which M_{ent} of ω_i is zero.

The fault isolation time is reported in milliseconds.

6.5 Results

Table 7 shows the main results of our experiments. As the full isolation requires a manual step (the selection of a user-command amongst the first lowest-entropy ones) and the simulation of the effect of the user-command we have shown only the first disambiguation step which leads to a lower diagnostic uncertainty.

All experiments were performed on a modern Pentium desktop (3 GHz), however, the implementation of the LYDIA-NG algorithms was single threaded. Further, we have used simulation databases, to improve the speed of the simulation process. The offline time for computing a simulation database is less than two hours. After that, the online step of looking-up the database and computing the residual takes less than 100 ms. These results show that LYDIA-NG can be used for real-time monitoring and diagnosis, processing telemetry data as it arrives, typically at a rate of 2 Hz.

The GOCE experiments showed no false-positive and false-negative scenarios, hence the diagnostic accuracy was one (see columns M_{fn}, M_{fp}, and M_{da} in Table 7). The classification errors and the isolation accuracy metrics are close to the optimal for the model, although the worst-case values of M_{err} and M_{ia} are not computed (i.e., the optimal values for the *worst-case* observation). The latter would be useful during the design stage and we plan to implement them in future work.

The relatively large memory consumption (511 MiB in all scenarios) is due to the use of non-compressed simulation databases. This is not a problem for ground (as opposed to on-board) as RAM for this kind of application is nowadays very cheap.

The fault isolation time M_{fi} is a worst-case estimation that includes the computation of disambiguation only. We assume that the user always chooses a command that does not lead to switching-off the faulty component (a user action that would masquerade the fault leads to a predicted uncertainty of zero and would be proposed by Alg. 2). Considering TST3, for example, the M_{fi} estimation is done assuming that the user switches-off all heater switches in the group before reaching the faulty one.

7 Conclusions

In this paper we have reported on the GENIUS project. The purpose of the project was to bring MBD one step closer to the real-world in assisting operators in satellite ground control.

In this project we have simulated telemetry that is close to real-world. We have defined a number of scenarios in which we have injected multiple, ambiguous, intermittent faults and we have defined a number of performance metrics with which we have tested diagnosis and disambiguation. Our approach is promising in that it delivers good scenario diagnostic accuracy, small number of classification errors. Further, the disambiguation approach we propose can reduce the isolation accuracy and assist operators in decision making and execution of (sequences of) telecommands.

References

[1] S. Narasimhan and L. Brownston, "Hyde–a general framework for stochastic and hybrid model-based diagnosis," in *Proc. 18th International Workshop on Principles of Diagnosis (DX'07), Nashville, USA*, 2007, pp. 162–169.

[2] M. Daigle, I. Roychoudhury, G. Biswas, X. Koutsoukos, A. Patterson-Hine, and S. Poll, "A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 40, no. 5, pp. 917–931, 2010.

[3] A. Zheng, I. Rish, and A. Beygelzimer, "Efficient test selection in active diagnosis via entropy approximation," *arXiv preprint arXiv:1207.1418*, 2012.

[4] S. Wu, Q. Yan, Y. Ren, and S. Guo, "Efficient probe prediction algorithm for fault diagnosis in computer networks," in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. IEEE, 2011, pp. 1–4.

[5] Y. Qiao, X. Qiu, L. Cheng, and L. Meng, "A methodology used to optimize probe selection for fault localization," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[6] G. Bellala, S. Bhavnani, and C. Scott, "Active diagnosis under persistent noise with unknown noise distribution: A rank-based approach," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[7] D. Tolpin and S. Shimony, "Rational value of information estimation for measurement selection," *arXiv preprint arXiv:1003.5305*, 2010.

[8] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman, "First international diagnosis competition - DXC'09," in *Proc. DX'09*, 2009, pp. 383–396.

[9] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos, "Advanced diagnostics and prognostics testbed," in *Proc. DX'07*, 2007, pp. 178–185.

[10] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, and A. van Gemund, "Empirical evaluation of diagnostic algorithm performance using a generic framework," *International Journal of Prognostics and Health Management*, pp. 1–28, 2010.

[11] A. Feldman and A. van Gemund, "A two-step hierarchical algorithm for model-based diagnosis," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06), Boston, Massachusetts, USA*, July 2006.

Table 7: Diagnostic metrics from the GOCE EPS experiments

Name	M_{fd} [ms]	M_{fn}	M_{fp}	M_{da} [ms]	M_{fi} [s]	M_{err}	M_{ia}	M_{ent}	M_{cpu} [ms]	M_{mem} [MiB]
TST1	–	0	0	1	–	0	289	0	114	511
TST2	88	0	0	1	–	0	289	0	88	511
TST3	88	0	0	1	505	1.45	287.55	0.004	88	511
TST4	89	0	0	1	–	0	289	0	89	511
TST5	90	0	0	1	–	1	288	0.0035	90	511
TST6	90	0	0	1	101	1	288	0.0016	90	511
TST7	93	0	0	1	505	3.96	285.04	0.0047	93	511
TST8	94	0	0	1	–	1	288	0.0022	94	511
TST9	84	0	0	1	–	1	288	0	84	511
TST10	–	–	–	–	–	–	–	–	–	–
TST11	77	0	0	1	505	1.83	287.17	0.0025	77	511

[12] J. de Kleer, A. Mackworth, and R. Reiter, "Characterizing diagnoses and systems," *Artificial Intelligence*, vol. 56, no. 2-3, pp. 197–222, 1992.

[13] A. Feldman, G. Provan, and A. van Gemund, "Approximate model-based diagnosis using greedy stochastic search," *Journal of Artificial Intelligence Research*, vol. 38, pp. 371–413, 2010.

[14] G. W. Roberts and A. S. Sedra, *SPICE*, ser. The Oxford Series in Electrical and Computer Engineering. Oxford University Press, 1996.

[15] S. A. P. S. H. M. Committee E-32, "Health and usage monitoring metrics, monitoring the monitor," Tech. Rep. ARP5783, February 2008.

[16] H. R. DePold, J. Siegel, and J. Hull, "Metrics for evaluating the accuracy of diagnostic fault detection systems," in *Proc. TURBO'04*, 2004.

[17] H. R. DePold, R. Rajamani, W. H. Morrison, and K. R. Pattipati, "A unified metric for fault detection and isolation in engines," in *Proc. TURBO'06*, 2006.

[18] C. E. Metz, "Basic principles of ROC analysis," *Nuclear Medicine*, vol. 8, no. 4, pp. 283–298, 1978.

[19] R. F. Orsagh, M. J. Roemer, C. J. Savage, and M. Lebold, "Development of performance and effectiveness metrics for gas turbine diagnostic technologies," in *Proc. AEROCONF'02*, vol. 6, 2002, pp. 2825–2834.

[20] M. Roemer, J. Dzakowic, R. F. Orsagh, C. S. Byington, and G. Vachtsevanos, "Validation and verification of prognostic health management technologies," in *Proc. AEROCONF'05*, 2005, pp. 3941–3947.

[21] M. Bartyś, R. Patton, M. Syfert, S. de las Heras, and J. Quevedo, "Introduction to the DAMADICS actuator FDI benchmark study," *Control Engineering Practice*, vol. 14, pp. 577–596, 2006.

Poster Papers

Diagnosis of Discrete-Event Systems with Stratified Behavior

Gianfranco Lamperti[1] and **Xiangfu Zhao**[2]
[1]Department of Information Engineering, University of Brescia, Italy
e-mail: lamperti@ing.unibs.it
[2]College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, China
e-mail: xiangfuzhao@zjnu.cn

Abstract

A gap still exists between the complexity of real discrete-event systems (DESs) and the effectiveness of state-of-the-art diagnosis techniques. To deal with this gap, a novel class of discrete-event systems, called higher-order DESs (HDESs) is introduced, along with a relevant diagnosis technique. The behavior of a HDES is stratified, resulting in a hierarchy of cohabiting sub-DESs, each one living its own life. The communication between subsystems at different levels relies on complex events, occurring when specific patterns of transitions are matched. Diagnosis of HDESs is context-sensitive and scalable.

1 Introduction

Diagnosis of DESs [Cassandras and Lafortune, 1999] has been attracting attention of the scientific community since the seminal work of [Sampath *et al.*, 1996]. However, a gap still exists between the complexity of real DESs and the effectiveness of state-of-the-art diagnosis techniques. A complex DES is not necessarily large (even though a large DES is likely to be complex). In our meaning, complexity refers to the mode in which the DES is organized, at different levels of abstraction, with each level being characterized by its proper behavior, which depends on the behaviors of lower-level layers, yet differs from just the composition of them. This property is called *behavior stratification*. In the literature, DESs are typically modeled as networks of interacting components, where the behavior of each component is described by a communicating automaton [Brand and Zafiropulo, 1983]. However, complexity of the DES has become a research issue only recently. Previous research has mainly focused on relevant yet different aspects, including incrementality [Baroni *et al.*, 1999; Grastien *et al.*, 2005], distribution/decentralization [Pencolé, 2000; Debouk *et al.*, 2000; Pencolé *et al.*, 2001; Debouk *et al.*, 2003; Grastien *et al.*, 2004; Pencolé and Cordier, 2005; Qiu and Kumar, 2006], and uncertainty/incompleteness [Lamperti and Zanella, 2002; Zhao and Ouyang, 2008; Lamperti and Zanella, 2011b; Kwong and Yonge-Mallo, 2011; Zhao *et al.*, 2012]. The notion of context-sensitive diagnosis has been introduced for DESs that are organized within abstraction hierarchies, so that candidate diagnoses can be generated at different abstraction levels [Lamperti and Zanella, 2011a]. Even in that work, albeit the diagnosis depends on the context, the DES is assumed to be a network of components with no behavior stratification. When behavior stratification occurs, the DES is called a *higher-order DES* (HDES).

2 Higher-Order Discrete-Event System

A higher-order DES, namely $\mathcal{H}$, is a tree where nodes are *components*. Leaf nodes are *basic components*, while internal nodes are *complex components*. The set of child components of a complex component X is indicated by $\mathcal{C}(X)$. Each (either basic or complex) component in $\mathcal{H}$ is defined in terms of a *topological model* and a *behavioral model*. The topological model consists of a set of *input terminals* and a set of *output terminals*. Components in $\mathcal{C}(X)$ are connected to one another through *links*, with each link exiting the output terminal of one component and entering the input terminal of another component. These connections form a network $\mathcal{N}(X)$. Let $\mathbf{I}$ and $\mathbf{O}$ denote the input and output terminals, respectively, of a component C. The behavioral model of C is a communicating automaton $(\mathbb{S}, \mathbb{I}, \mathbb{O}, \mathbb{T})$, where $\mathbb{S}$ is the set of states, $\mathbb{I}$ the set of input events, $\mathbb{O}$ the set of output events, and $\mathbb{T} : \mathbb{S} \times (\mathbb{I} \times \mathbf{I}) \times 2^{(\mathbb{O}\times\mathbf{O})} \mapsto 2^{\mathbb{S}}$ the (nondeterministic) transition function. As such, a transition is triggered by an input event and generates a (possibly empty) set of output events. The latters are thus made available as input events at the corresponding component terminals, while the input (triggering) event is consumed. A transition can be triggered only if all links, towards which output events are generated, are empty (no event is in the link). Each complex component X is endowed with a *Cot* additional (virtual) input terminal, which is sensitive to *complex events*. A complex event is a set of *pattern events*. A pattern event occurs when the network $\mathcal{N}(X)$ undergoes a string of transitions matching a given regular expression. The alphabet of such a regular expression is the whole set of transitions of components in $\mathcal{C}(X)$. In general, for each complex component X, a set $\mathcal{P}(X)$ of *patterns* is defined, with each pattern being a pair (p, r), where p is the name of a pattern event and r a regular expression on transitions of $\mathcal{C}(X)$. Several pattern events may occur simultaneously. In fact, given a string $\mathcal{T}$ of transitions of components in $\mathcal{C}(X)$, each suffix of $\mathcal{T}$ matching a regular expression in $\mathcal{P}(X)$ gives rise to a pattern event. The whole set of these pattern events forms a complex event.

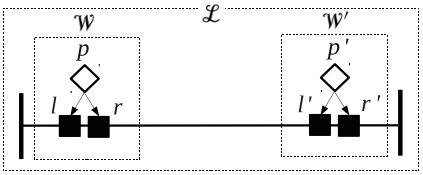


Figure 1: HDES: protected power transmission line.

Example 1. Shown in Fig. 1 is a HDES representing a power transmission line $\mathcal{L}$. On both sides, the line is pro-

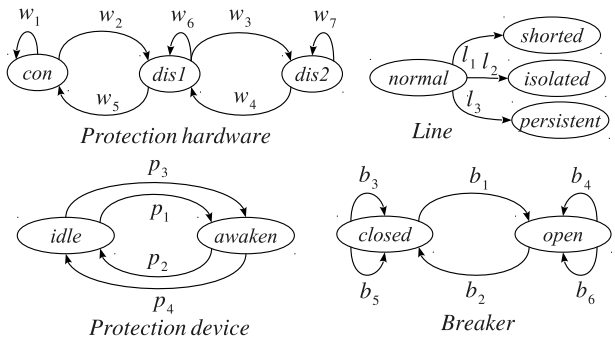


Figure 2: Behavioral models.

tected from short circuits by a protection hardware, $\mathcal{W}$ and $\mathcal{W}'$, each composed of a protection device, p and p', respectively, and two breakers, l and r, and l' and r', respectively. Boxes denote complex components $\mathcal{L}$, $\mathcal{W}$, and $\mathcal{W}'$. We assume that the output terminal of the protection device is exited by two links directed to the input terminals of the two breakers. The protection device is sensitive to short circuits on the line, detected as lowering of voltage, in which case it commands the breakers to open in order to isolate the line (just one open breaker on both sides is sufficient for isolation). Once the line is isolated, the short circuit is expected to vanish. If so, the protection device commands both breakers to close in order to reconnect the line (all breakers need to be closed). However, faulty behavior may occur:

- The protection device sends the wrong command;
- The breaker does not react to the command of protection device (thereby remaining in its state);
- The protection hardware fails to either disconnect or connect the line (if just one breaker does not open, the protection hardware is normal, as disconnection occurs; for the connection, both breakers must close);
- The line is not isolated, or once the short circuit has vanished, the line is not reconnected, or after reconnection the short circuit still persists (e.g. a tree fallen on the line).

Table 1: Details for transitions of models in Fig. 2.

T	*Action performed by component transition T*
p_1	Detects low voltage and outputs *op* (open) event
p_2	Detects normal voltage and outputs *cl* (close) event
p_3	Detects low voltage, yet outputs *cl* event
p_4	Detects normal voltage, yet outputs *op* event
b_1	Consumes *op* event and opens
b_2	Consumes *cl* event and closes
b_3	Consumes *op* event, yet keeps closed
b_4	Consumes *cl* event, yet keeps open
b_5	Consumes *cl* event
b_6	Consumes *op* event
w_1	Consumes **nd** (not disconnected) complex event
w_2, w_3	Consumes **di** (disconnected) complex event
w_4, w_5	Consumes **co** (connected) complex event
w_6, w_7	Consumes **nc** (not connected) complex event
l_1	Consumes **ni** (not isolated) complex event
l_2	Consumes **nr** (not reconnected) complex event
l_3	Consumes **ps** (persistent short) complex event

Outlined in Fig. 2 are the behavioral models. Details on component transitions are provided in Table 1. Patterns relevant to complex events in Table 1 are defined in Table 2. For instance, consider pattern event *ps* (persistent short circuit) and the corresponding regular expression, where $*$ means repetition zero or more times, $+$ means repetition one or more times, and $\neg$ (negation) means any transition different from its argument. Event *ps* occurs when either $\mathcal{W}$ or $\mathcal{W}'$ repeats one or more times the following sequence of transitions: it closes (w_5) and then, after zero or more occurrences of w_1, it opens again (w_2), followed by zero or more transitions other than w_5. Notice that *ps* is a single pattern event, while **ps** is the singleton $\{ps\}$ (complex event).[1] ◇

2.1 Pattern Space

In order to detect complex events, the state of the matching of patterns is to be maintained somewhere. To this end:

- For each pattern (p, r), a deterministic *pattern automaton* A equivalent to regular expression r is generated, where final states are marked by pattern event p.
- For each complex component X for which the set $\{A_1, \ldots, A_k\}$ of pattern automata were generated, a *pattern space*, written $Pts(X)$, is created as follows:
  1. A nondeterministic automaton N is created by generating initial state S_0 and one empty transition from S_0 to each initial state of A_i, $i \in [1\,..\,k]$;
  2. In each A_i, $i \in [1\,..\,k]$, an empty transition from each non-initial state to S_0 is inserted;[2]
  3. N is determinized into $Pts(X)$, where each final state S is marked by the union **p** of the pattern events that are associated with the states in S that are final in the corresponding pattern automaton.[3]

Table 2: Specification of patterns by regular expressions.

Pattern event	*Regular expression*
di	$b_1(l) \mid b_1(r)$
co	$b_2(l) \mid b_2(r)$
nd	$p_3(p) \mid p_1(p)((b_3(l)b_3(r)) \mid (b_3(r)b_3(l)))$
nc	$p_4(p) \mid p_2(p)(b_5(r)?\, b_4(l) \mid b_5(l)?\, b_4(r))$
ni	$w_1(\mathcal{W}) \mid w_1(\mathcal{W}')$
nr	$w_6(\mathcal{W}) \mid w_7(\mathcal{W}) \mid w_6(\mathcal{W}') \mid w_7(\mathcal{W}')$
ps	$(w_5(\mathcal{W})w_1(\mathcal{W})^*w_2(\mathcal{W})(\neg w_5(\mathcal{W}))^*)^+ \mid$ $(w_5(\mathcal{W}')w_1(\mathcal{W}')^*w_2(\mathcal{W}')(\neg w_5(\mathcal{W}'))^*)^+$

Example 2. Based on Example 1, consider complex component $\mathcal{W}$, whose patterns are defined in Table 2 (top). Following the steps specified above, $Pts(\mathcal{W})$ is generated as detailed in Table 3. The main part of the table represents the transition function, where for each component transition $T \in \{p_1(p), \ldots, b_5(r)\}$ (listed in the first column), and for each state $\mathcal{P}_i$, $i \in [0\,..\,10]$ (listed in the first row), the reached state is indicated in the cell $(T, \mathcal{P}_i)$. Moreover, highlighted states are final, namely $\mathcal{P}_1$, $\mathcal{P}_4$, $\mathcal{P}_7$, $\mathcal{P}_8$. Complex events associated with final states are listed in the last row, namely **di**, **co**, **nc**, and **nd** (details are in Table 1). These are singletons of the homonymous pattern event. ◇

Proposition 1. *The set* **p** *marking a final state* S_f *of* $Pts(X)$ *is composed of the pattern events* p *such that* $(p, r) \in \mathcal{P}(X)$, $\mathcal{T}$ *is a string in the language of* $Pts(X)$ *ending at* S_f, $\mathcal{T}'$ *is a string matching regular expression* r, *and* $\mathcal{T}'$ *is a suffix of* $\mathcal{T}$.

Proof. (*Sketch*) In creating $Pts(X)$, if we omit step 2 then the language of $Pts(X)$ will be the union of the languages of regular expressions involved in $\mathcal{P}(X)$, where each string

[1] Although a complex event is a set of pattern events, incidentally in our example all complex events are singletons.

[2] This allows for pattern-matching of overlapping strings.

[3] Each state S of the deterministic automaton is identified by a subset of the states of the equivalent nondeterministic automaton.

Table 3: Tabular specification of pattern space $Pts(\mathcal{W})$.

$T \setminus \mathcal{P}_i$	$\mathcal{P}_0$	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{P}_5$	$\mathcal{P}_6$	$\mathcal{P}_7$	$\mathcal{P}_8$	$\mathcal{P}_9$	$\mathcal{P}_{10}$
$p_1(p)$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$	$\mathcal{P}_2$
$p_2(p)$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$	$\mathcal{P}_3$
$p_3(p)$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$	$\mathcal{P}_8$
$p_4(p)$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$	$\mathcal{P}_7$
$b_1(l)$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$
$b_1(r)$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$	$\mathcal{P}_1$
$b_2(l)$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$
$b_2(r)$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$	$\mathcal{P}_4$
$b_3(l)$	–	–	$\mathcal{P}_5$	–	–	–	$\mathcal{P}_8$	–	–	–	–
$b_3(r)$	–	–	$\mathcal{P}_6$	–	–	$\mathcal{P}_8$	–	–	–	–	–
$b_4(l)$	–	–	–	$\mathcal{P}_7$	–	–	–	–	–	–	$\mathcal{P}_7$
$b_4(r)$	–	–	–	$\mathcal{P}_7$	–	–	–	–	–	$\mathcal{P}_7$	–
$b_5(l)$	–	–	–	$\mathcal{P}_9$	–	–	–	–	–	–	–
$b_5(r)$	–	–	–	$\mathcal{P}_{10}$	–	–	–	–	–	–	–
		di			**co**			**nc**	**nd**		

ending at S_f matches the regular expression associated with a pattern event in **p**. Consequently, the statement of the theorem should be restricted in the last condition by $\mathcal{T}' = \mathcal{T}$. The more relaxed condition, namely $\mathcal{T}'$ being a suffix of $\mathcal{T}$, comes from step 2 of the construction: in general, because of additional empty transitions, each string $\mathcal{T}$ ending at S_f matches regular expressions only in its suffixes. □

2.2 Behavior Space

Starting from its initial state $\mathcal{H}_0$, HDES $\mathcal{H}$ may perform a sequence of component transitions within its *behavior space*, written $Bsp(\mathcal{H}, \mathcal{H}_0)$, which is a finite automaton

$$Bsp(\mathcal{H}, \mathcal{H}_0) = (\mathbf{S}, \mathbf{T}, S_0).$$

S is the set of states $(\mathcal{S}, \mathcal{E}, \mathcal{P})$, with $\mathcal{S} = (s_1, \ldots, s_n)$ being the tuple of states of (both basic and complex) components in $\mathcal{H}$. $\mathcal{E} = (e_1, \ldots, e_m)$ is the tuple of events at input terminals of components in $\mathcal{H}$ (ϵ indicates no event), and $\mathcal{P} = (P_1, \ldots, P_k)$ the tuple of pattern-space states. $S_0 = (\mathcal{H}_0, \mathcal{E}_0, \mathcal{P}_0)$ is the initial state, where $\mathcal{E}_0 = (\epsilon, \ldots, \epsilon)$ and $\mathcal{P}_0 = (P_{10}, \ldots, P_{k0})$ the tuple of the initial states of pattern spaces $Pts(X_1), \ldots, Pts(X_k)$, respectively. **T** is the transition function, where

$$(\mathcal{S}, \mathcal{E}, \mathcal{P}) \xrightarrow{T} (\mathcal{S}', \mathcal{E}', \mathcal{P}') \in \mathbf{T} \text{ if and only if:}$$

1. $T = s \xrightarrow{(e,I) \mid E_\mathrm{out}} s'$ (where e is the input event and E_out the output events with relevant terminals) is a transition of a (either basic or complex) component C such that s equals one element of $\mathcal{S}$, I is an input terminal of C, and $\mathcal{E}(I) = e$;
2. $\mathcal{S}'$ differs from $\mathcal{S}$ only in s' replacing s;
3. If $C \in \mathcal{C}(X_i), i \in [1\,..\,k]$, then $\mathcal{P}'$ differs from $\mathcal{P}$ only in the i-th element as follows:
$$\mathcal{P}'(P_i) = \begin{cases} \bar{P} & \text{if } \mathcal{P}(P_i) \xrightarrow{T} \bar{P} \in Pts(X_i) \\ P_{i0} & \text{otherwise;} \end{cases}$$
4. $\mathcal{E}'$ differs from $\mathcal{E}$ based on these conditions:
   (*a*) $\mathcal{E}'(I) = \epsilon$ (event e is consumed);
   (*b*) $\forall (o, O) \in E_\mathrm{out}$, denoting with I' the terminal entered by the link exiting O, we have: $\mathcal{E}(I') = \epsilon$ (no event is initially present at terminal I') and $\mathcal{E}'(I') = o$ (o is then present at terminal I');
   (*c*) If $\mathcal{P}'(P_i) \neq \mathcal{P}(P_i)$, $i \in [1\,..\,k]$, $\mathcal{P}'(P_i)$ is final in $Pts(X_i)$ and marked by complex event **p**, then $\mathcal{E}(Cot) = \epsilon$, $\mathcal{E}'(Cot) = \mathbf{p}$.

As such, transitions in $Bsp(\mathcal{H}, \mathcal{H}_0)$ are marked by transitions of components in $\mathcal{H}$. The new state not only reflects the consumption of input event e of the component transition T and the generation of the output events in E_out: it also accounts for the possible occurrence of a complex event **p**.

A string in the language of $Bsp(\mathcal{H}, \mathcal{H}_0)$ is a *history* of $\mathcal{H}$. The behavior space is defined for formal reasons only, as its actual materialization is impractical in real HDESs.

3 Diagnosis Problem

Diagnosing a HDES means finding the faults in its history. A history can be observed only in its observable transitions, as a sequence of *observation labels*, called the *trace* of the history, with each label being associated with an observable transition. The diagnosis process is complicated by two facts. First, several histories may generate the same trace. Second, because of noise and distribution of the channels conveying labels from the HDES, rather than a sequence of labels, the trace is perceived as a DAG, called *temporal observation*, where each node contains a set of observation labels and each arc represents partial (rather than total) temporal ordering between observation labels. Consequently, several *candidate traces* are observed, each one made up by choosing a label in each node of the DAG fulfilling the partial ordering imposed by arcs. Furthermore, since several (even infinite) histories may be consistent with the same trace, the diagnosis output is a set of *candidate diagnoses*, with each candidate corresponding to a subset of the possible histories. However, despite the possible infinite set of histories consistent with the temporal observation, the set of candidate diagnoses is always finite (being it bounded from above by the powerset of component transitions).

A *diagnosis problem* for a HDES $\mathcal{H}$ is a quadruple

$$\wp(\mathcal{H}) = (\mathcal{H}_0, \mathcal{V}, \mathcal{O}, \mathcal{R}), \text{ where}$$

- $\mathcal{H}_0$ is the initial state of $\mathcal{H}$;
- $\mathcal{V}$ is the *viewer* of $\mathcal{H}$, a set of pairs (T, ℓ), where T is a component transition and ℓ an observation label, with $h_{[\mathcal{V}]}$ denoting the trace of history h based on $\mathcal{V}$;
- $\mathcal{O}$ is the *temporal observation* of $\mathcal{H}$, with $\|\mathcal{O}\|$ denoting the set of candidate traces in $\mathcal{O}$;
- $\mathcal{R}$ is the *ruler* of $\mathcal{H}$, a set of pairs (T, f), where T is a component transition and f a fault label, with $h_{[\mathcal{R}]}$ denoting the *diagnosis* of history h based on $\mathcal{R}$, defined as:
$$h_{[\mathcal{R}]} = \{f \mid T \in h, (T, f) \in \mathcal{R}\}.$$

If a transition T is included in $\mathcal{V}$, then it is *observable*, otherwise it is *unobservable*. If T is included in $\mathcal{R}$, then it is *faulty*, otherwise it is *normal*.

The *solution* Δ of $\wp(\mathcal{H})$ is the set of candidate diagnoses:

$$\Delta(\wp(\mathcal{H})) = \left\{\delta \mid h \in Bsp(\mathcal{H}, \mathcal{H}_0), h_{[\mathcal{V}]} \in \|\mathcal{O}\|, \delta = h_{[\mathcal{R}]}\right\}.$$

Each candidate diagnosis is the set of faulty transitions of a history that is consistent with the temporal observation.

For practical reasons, instead of processing observation $\mathcal{O}$, the *index space* of $\mathcal{O}$ is generated [Lamperti and Zanella, 2002], namely $Isp(\mathcal{O})$. This is a deterministic automaton whose language equals $\|\mathcal{O}\|$ (the set of candidate traces).

Example 3. With reference to Example 1, we define the diagnostic problem for the left-hand side protection-hardware as $\wp(\mathcal{W}) = (\mathcal{W}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$, where:

- In $\mathcal{W}_0$ both breakers are *closed*, protection device is *idle*, and protection hardware is *con* (see Fig. 2);
- $\mathcal{V} = \{(b_1(l), opl), (b_1(r), opr), (b_2(l), cll), (b_2(r), clr), (p_1, awk), (p_2, ide), (p_3, awk), (p_4, ide)\}$;
- $\mathcal{O}$ is the temporal observation displayed in Fig. 3 (left);

- $\mathcal{R} = \{(b_3(l), nol), (b_3(r), nor), (b_4(l), ncl), (b_4(r), ncr), (p_3, fop), (p_4, fcp), (w_1, fdw), (w_6, fcw), (w_7, fcw)\}, (l_1, fil), (l_2, frl), (l_3, psl)\}$.

Temporal observation $\mathcal{O}$ (Fig. 3, left) includes four nodes, with ω_1 and ω_4 containing two observation labels (ϵ is the empty label). As such, $\mathcal{O}$ is uncertain. Because of this uncertainty and partial temporal ordering, $\mathcal{O}$ embodies six candidate traces, which are the strings of the language of the index space $Isp(\mathcal{O})$ displayed on the right of Fig. 3 (where $\Im_3$ and $\Im_5$ are final). ◇

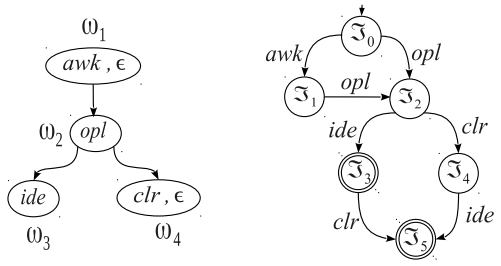


Figure 3: Temporal observation $\mathcal{O}$ (left) and $Isp(\mathcal{O})$ (right).

4 Diagnosis Computation

The definition of diagnosis-problem solution is not operational in nature: it refers to the behavior space, which is assumed not to be available in practice. The diagnosis engine is expected to be sound and complete in generating the solution of the problem, without the availability of the behavior space. To this end, it reconstructs only the subpart of the behavior space that is consistent with the temporal observation. In doing so, the reconstruction needs to keep four sorts of information: the state of components, the state of input terminals, the state of the matching of pattern events, and the state of the matching of the temporal observation. Specifically, the solution of a diagnostic problem $\wp(\mathcal{H}) = (\mathcal{H}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$ is computed in three steps:

- Generating the *index space* of temporal observation $\mathcal{O}$;
- Generating the subspace of $Bsp(\mathcal{H}, \mathcal{H}_0)$ that is consistent with temporal observation $\mathcal{O}$, based on viewer $\mathcal{V}$, called the *behavior* of $\wp(\mathcal{H})$, written $Bhv(\wp(\mathcal{H}))$;
- Decorating the states of behavior $Bhv(\wp(\mathcal{H}))$ by the associated set of candidate diagnoses.

The actual solution $\Delta(\wp(\mathcal{H}))$ is the union of the decorations associated with final states of $Bhv(\wp(\mathcal{H}))$ (see Theorem 1).

Formally, $Bhv(\wp(\mathcal{H}))$ is defined as follows. Let $\mathbf{S}$ be the domain of tuples $(s_1, \ldots, s_n)$ of states of components in $\mathcal{C}(\mathcal{H})$. Let $\mathbf{E}$ be the domain of tuples $(e_1, \ldots, e_m)$ of events at input terminals (other than *Ext*) of components in $\mathcal{C}(\mathcal{H})$. Let $\mathfrak{I}$ be the domain of states in $Isp(\mathcal{O})$. Let $\boldsymbol{\mathcal{P}}$ be the domain of tuples $(P_1, \ldots, P_k)$ of pattern-space states. The behavior of $\wp(\mathcal{H})$ is a deterministic automaton:

$$Bhv(\wp(\mathcal{H})) = (\mathbb{S}, \mathbb{T}, S_0, \mathbb{S}_f), \text{where}$$

- $\mathbb{S} \subseteq \mathbf{S} \times \mathbf{E} \times \boldsymbol{\mathcal{P}} \times \mathfrak{I}$ is the set of states;
- $S_0 = (\mathcal{H}_0, \mathcal{E}_0, \mathcal{P}_0, \Im_0)$ is the initial state, where $\mathcal{E}_0 = (\epsilon, \ldots, \epsilon)$, $\mathcal{P}_0 = (P_{10}, \ldots, P_{k0})$ the tuple of the initial states of pattern spaces $Pts(X_1), \ldots, Pts(X_k)$, respectively, and $\Im_0$ the initial state of $Isp(\mathcal{O})$;
- $\mathbb{S}_f = \{(\mathcal{S}, \mathcal{E}, \mathcal{P}, \Im) \mid \mathcal{E} = (\epsilon, \ldots, \epsilon), \Im \text{ is final } \}$ is the set of final states;
- $\mathbb{T}$ is the transition function, where

$$(\mathcal{S}, \mathcal{E}, \mathcal{P}, \Im) \xrightarrow{T} (\mathcal{S}', \mathcal{E}', \mathcal{P}', \Im') \in \mathbb{T} \text{ if and only if:}$$

1. Conditions 1–4 on $\mathcal{S}'$, $\mathcal{E}'$, and $\mathcal{P}'$, for the transition function of $Bsp(\mathcal{H}, \mathcal{H}_0)$, hold;
2. $\Im' = \begin{cases} \bar{\Im} & \text{if } (T, o) \in \mathcal{V}, \Im \xrightarrow{o} \bar{\Im} \in Isp(\mathcal{O}) \\ \Im & \text{otherwise.} \end{cases}$

The actual algorithm that builds $Bhv(\wp(\mathcal{H}))$ starts from the initial state S_0 and generates all possible transitions based on the conditions above. Eventually, it removes all spurious states and transitions that are not in a path from the initial state to a final state.

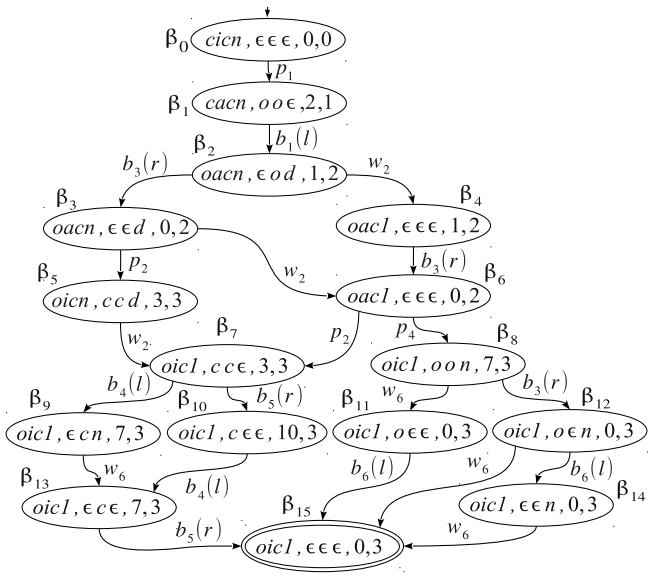


Figure 4: Behavior $Bhv(\wp(\mathcal{W}))$.

Example 4. With reference to $\wp(\mathcal{W})$ in Example 3, shown in Fig. 4 is $Bhv(\wp(\mathcal{W}))$, including states $\beta_0, \ldots, \beta_{15}$, with β_{15} final. Each state $(\mathcal{S}, \mathcal{E}, \mathcal{P}, \Im)$ is such that $\mathcal{S}$ is the quadruple of states for l, p, r, and $\mathcal{W}$, where *closed*, *open*, *idle*, *awaken*, *con*, and *dis1* are written c, o, i, a, n, and 1, respectively, $\mathcal{E}$ is the triple of events at input terminals of l, r, and $\mathcal{W}$, respectively, where *op*, *cl*, **di**, and **nc** are written o, c, d, and n, respectively, while $\mathcal{P}$ and $\Im$ are the indices of states in $Pts(w)$ and $Isp(\mathcal{O})$, respectively. For instance, $\beta_2 = (oacn, \epsilon od, 1, 2)$ stands for $\mathcal{S} = (open, awaken, closed, con)$, $\mathcal{E} = (\epsilon, op, \mathbf{di})$, $\mathcal{P} = \mathcal{P}_1$, and $\Im = \Im_2$. ◇

Once the behavior has been constructed, each state S of $Bhv(\wp(\mathcal{H}))$ is decorated by the minimal set of candidate diagnoses $\Delta(S)$ fulfilling the following two inductive rules:

(1) For the initial state: $\Delta(S_0) = \{\emptyset\}$.

(2) For each transition $S \xrightarrow{T} S'$ in $Bhv(\wp(\mathcal{H}))$:
If T is normal then $\delta \in \Delta(S) \Rightarrow \delta \in \Delta(S')$;
If $(T, f) \in \mathcal{R}$ then $\delta \in \Delta(S) \Rightarrow (\delta \cup \{f\}) \in \Delta(S')$.

The algorithm that decorates $Bhv(\wp(\mathcal{H}))$ starts by applying the first rule, marking the initial state with the singleton of the empty diagnosis. Then, based on the decoration of the initial state, it continuously applies the second rule for each transition exiting a state S whose decoration has changed. If $(T, f) \in \mathcal{R}$ then T is faulty, with f being the relevant fault. If so, each candidate diagnosis in $\Delta(S)$ extended by fault f is also a candidate diagnosis in $\Delta(S')$. Instead, if T is normal, all candidate diagnoses in $\Delta(S)$ are candidate diagnoses in $\Delta(S')$ too. As such, $\Delta(S)$ is constructed as the set of diagnoses relevant to histories ending at state S. The algorithm terminates when the application of the second rule does not cause any change in any decoration.

Table 4: Decoration of $Bhv(\wp(\mathcal{W}))$.

States	*Decoration*
$\beta_0, \beta_1, \beta_2, \beta_4$	$\{\emptyset\}$
$\beta_3, \beta_5, \beta_6, \beta_7, \beta_{10}$	$\{\{nor\}\}$
β_9	$\{\{nor, ncl\}\}$
β_{13}	$\{\{nor, ncl, fcw\}\}$
$\beta_8, \beta_{12}, \beta_{14}$	$\{\{nor, fcp\}\}$
β_{11}	$\{\{nor, fcp, fcw\}\}$
β_{15}	$\{\{nor, ncl, fcw\}, \{nor, fcp, fcw\}\}$

Example 5. Based on ruler $\mathcal{R}$ (Example 3), the behavior in Fig. 4 will be decorated as specified in Table 4. Therefore, two candidate diagnoses are associated with final state β_{15}, namely $\delta_1 = \{nor, ncl, fcw\}$, and $\delta_2 = \{nor, fcp, fcw\}$, corresponding to these two scenarios:

δ_1 : Breaker r fails to open, breaker l fails to close, and protection hardware $\mathcal{W}$ fails to connect;

δ_2 : Breaker r fails to open, protection device trips breakers to open rather than to close, and $\mathcal{W}$ fails to connect.

Based on Theorem 1, $\{\delta_1, \delta_2\}$ is the solution of $\wp(\mathcal{W})$. Albeit we have two candidates, since $\delta_1 \cap \delta_2 = \{nor, fcw\}$, certainly r failed to open and $\mathcal{W}$ failed to connect. ◇

Theorem 1. *Let $\wp(\mathcal{H}) = (\mathcal{H}_0, \mathcal{V}, \mathcal{O}, \mathcal{R})$. Let $\mathcal{B}^s$ and $\mathcal{B}^v$ denote $Bsp(\mathcal{H}, \mathcal{H}_0)$ and $Bhv(\wp(\mathcal{H}))$, respectively. Let $\Delta(\mathcal{B}^v)$ denote the union of the sets of diagnoses decorating the final states of $\mathcal{B}^v$. Then, $\Delta(\mathcal{B}^s) = \Delta(\mathcal{B}^v)$.*

Proof. (*Sketch*) Grounded on Lemmas 1.1–1.5.

Lemma 1.1. *If history $h \in \mathcal{B}^v$ then $h \in \mathcal{B}^s$.*

This derives from the fact B^v differs from $\mathcal{B}^s$ in field $\Im$, which is irrelevant for conditions on $\mathcal{S}'$, $\mathcal{E}'$, and $\mathcal{P}'$. By induction on h, starting from the initial state, each new transition applicable in $\mathcal{B}^v$ is applicable in $\mathcal{B}^s$ too.

Lemma 1.2. *If history $h \in \mathcal{B}^v$ then $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$.*

Recall that $h_{[\mathcal{V}]}$ is the sequence of observable labels associated with visible transitions in viewer $\mathcal{V}$. Based on the definition of $\mathcal{B}^v$, $h_{[\mathcal{V}]}$ belongs to the language of $Isp(\mathcal{O})$, which equals $\|\mathcal{O}\|$. Thus, $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$.

Lemma 1.3. *If history $h \in \mathcal{B}^s$, $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$, then $h \in \mathcal{B}^v$.*

By induction on h, starting from the initial state, each new transition T applicable in $\mathcal{B}^s$ is applicable in $\mathcal{B}^v$ too. In fact, if T is invisible, no further condition is required. If T is visible, based on the assumption $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ and on the fact that the language of $Isp(\mathcal{O})$ equals $\|\mathcal{O}\|$, the label associated with T in viewer $\mathcal{V}$ matches a transition in $Isp(\mathcal{O})$.

Lemma 1.4. *If history $h \in \mathcal{B}^v$ ends at final state S_f then $\Delta(S_f)$ includes a candidate diagnosis $h_{[\mathcal{R}]}$.*

Based on the two rules for decoration of $\mathcal{B}^v$, by induction on h, starting from the initial state (h empty) and the empty diagnosis δ, the addition of a new transition T in h extends δ (within the decoration of the new state) by either nothing (T normal) or a fault label (T faulty). Upon the last transition of h, δ includes all fault labels associated with faulty transition in ruler $\mathcal{R}$, in other words $\delta = h_{[\mathcal{R}]}$.

Lemma 1.5. *If S_f is a final state in $\mathcal{B}^v$ and $\delta \in \Delta(S_f)$ then $\exists$ history $h \in \mathcal{B}^v$ ending at S_f such that $h_{[\mathcal{R}]} = \delta$.*

Based on the decoration rules for $\mathcal{B}^v$, δ is incrementally generated starting from the empty diagnosis initially associated with S_0, by inserting each faulty label associated with each faulty transition encountered in a path from S_0 to S_f. This path is a history provided that it is finite. In fact, cycles in $\mathcal{B}^v$ allow for an infinite number of applications of the second decoration rule. However, since δ is a set, once a cycle has been covered, all associated fault labels are inserted into δ. Successive iterations of the cycle do not extend δ because of duplicate removals. Thus, δ can always be generated by a finite history h, in other words, $\delta = h_{[\mathcal{R}]}$.

To prove Theorem 1, we show $\delta \in \Delta(\mathcal{B}^v) \Leftrightarrow \delta \in \Delta(\mathcal{B}^s)$. On the one hand, if $\delta \in \Delta(\mathcal{B}^v)$ then, based on Lemmas 1.1, 1.2, and 1.5, there exists a history $h \in \mathcal{B}^s$ such that $h_{[\mathcal{V}]} \in \|\mathcal{O}\|$ and $h_{[\mathcal{R}]} = \delta$, that is, $\delta \in \Delta(\mathcal{B}^s)$. On the other, if $\delta \in \Delta(\mathcal{B}^s)$ then, based on Lemmas 1.3 and 1.4, there exists a history $h \in \mathcal{B}^v$ ending at final state S_f such that $\delta = h_{[\mathcal{R}]}$ and $\delta \in \Delta(S_f)$, that is, $\delta \in \Delta(\mathcal{B}^v)$. □

5 Discussion

HDESs are a means of modeling complex DESs, where behavior is stratified and events can be generated by patterns of transitions. In spite of being influenced by other components, each internal node X of the hierarchy is a complex component living its own life. This means that X has its own behavioral model, which does not coincide with the composition of the behavioral models of its components. This results in a hierarchical system (HDES) made up of several cohabiting subsystems accommodated at different abstraction levels. The diagnosis technique defined for HDESs is model-based in nature: diagnosis is output based on the model of the system and the temporal observation. Only the portion of the behavior space consistent with the observation is reconstructed and eventually decorated by candidate diagnoses. Not only does separation of concerns apply to the modeling, it also applies to the diagnosis task. Since each (complex) component is provided with its own behavioral model, diagnosis is context-sensitive [Lamperti and Zanella, 2011a]. Moreover, depending on the degree of constraints on computational resources and time response of the diagnosis engine, model-based reasoning can be scaled to a convenient level of abstraction. This means restricting the HDES $\mathcal{H}$ to a portion $\mathcal{H}'$ (for instance, a complex component along with its children) and projecting the temporal observation $\mathcal{O}$ on $\mathcal{O}'$, resulting from the removal of irrelevant labels, nodes, and arcs. This way, within the context of $\mathcal{H}'$, the diagnosis output is complete, even if not sound (due to the removal of the behavioral constraints imposed by $\mathcal{H} - \mathcal{H}'$ and the observation constraint imposed by $\mathcal{O} - \mathcal{O}'$). To refine diagnosis, both $\mathcal{H}'$ and $\mathcal{O}'$ may be then enlarged to a suitable extent.

6 Related Work

This paper substantially extends the idea of context-sensitive diagnosis [Lamperti and Zanella, 2011a] in three directions. First, in [Lamperti and Zanella, 2011a] pattern stratification is only apparent, as, after macro-substitution, the regular expression is invariably defined on (basic) component transitions. In this paper, pattern stratification is real, since regular expressions are defined on the transitions of possibly complex components. Second, in [Lamperti and Zanella, 2011a] faults are associated with pattern matching. In this paper, faults are associated with transitions of components: pattern matching generates pattern events and, by union, complex events, to which complex components are sensitive. More importantly, in [Lamperti and Zanella, 2011a] context-sensitivity is defined on active systems, while this paper deals with HDESs, which provide behavioral stratification: separation of concerns holds not only for diagnosis but also for behavior. This paper also differs from [Jéron *et al.*, 2006], where the notion of supervision pattern is introduced, mainly because neither a hierarchical

structure for the system is conceived nor behavioral stratification is applicable. HDESs are not HFSMs (*Hierarchical Finite State Machines*). The notion of HFSM was inspired by statecharts [Harel, 1987], a visual formalism for complex systems. The most important feature of a HFSM is hierarchical state-nesting: if a system is in a nested state (substate), it is also in all its surrounding states (superstates). Moreover, transitions are defined at each level of the hierarchy. A simplified version of statechart, namely HFSM, was considered for solving a class of control problems in [Brave and Heymann, 1993]. Recently, diagnosis of HFSMs has been considered in [Idghamishi and Zad, 2004; Paoli and Lafortune, 2008]. However, no patterns are involved, events are simple, and diagnosis is context-free.

7 Conclusion

HDESs are a means to formalize complex DESs with behavior stratification. This allows for the modeling of a hierarchy wherein different, yet integrated, subsystems coexist, each one living its own life. This also allows for context-sensitivity and scalability of diagnosis.

Acknowledgment

This work was supported in part by NSFC under Grant No. 61003101, and Zhejiang Provincial Natural Science Foundation under Grant No. Y1100191.

References

[Baroni *et al.*, 1999] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183, 1999.

[Brand and Zafiropulo, 1983] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of ACM*, 30(2):323–342, 1983.

[Brave and Heymann, 1993] H. Brave and M. Heymann. Control of discrete event sysytems modeled as hierarchical state machines. *IEEE Transactions on Automatic Control*, 38(12):1803–1819, 1993.

[Cassandras and Lafortune, 1999] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, volume 11 of *The Kluwer International Series in Discrete Event Dynamic Systems*. Kluwer Academic Publishers, Boston, MA, 1999.

[Debouk *et al.*, 2000] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 10:33–86, 2000.

[Debouk *et al.*, 2003] R. Debouk, S. Lafortune, and D. Teneketzis. On the effect of communication delays in failure diagnosis of decentralized discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 13:263–289, 2003.

[Grastien *et al.*, 2004] A. Grastien, M.O. Cordier, and C. Largouët. Extending decentralized discrete-event modelling to diagnose reconfigurable systems. In *Fifteenth International Workshop on Principles of Diagnosis –DX'04*, pages 75–80, Carcassonne, F, 2004.

[Grastien *et al.*, 2005] A. Grastien, M.O. Cordier, and C. Largouët. Incremental diagnosis of discrete-event systems. In *Sixteenth International Workshop on Principles of Diagnosis DX'05*, pages 119–124, Monterey, CA, 2005.

[Harel, 1987] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Idghamishi and Zad, 2004] A.M. Idghamishi and S.H. Zad. Fault diagnosis in hierarchical discrete-event systems. In *43rd IEEE Conference on Decision and Control*, pages 63–68, Paradise Island, BSH, 2004.

[Jéron *et al.*, 2006] T. Jéron, H. Marchand, S. Pinchinat, and M.O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Seventeenth International Workshop on Principles of Diagnosis DX'06*, pages 117–124, Peñaranda de Duero, E, 2006.

[Kwong and Yonge-Mallo, 2011] R.H. Kwong and D.L. Yonge-Mallo. Fault diagnosis in discrete-event systems: incomplete models and learning. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 41(1):118–130, 2011.

[Lamperti and Zanella, 2002] G. Lamperti and M. Zanella. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, 137(1–2):91–163, 2002.

[Lamperti and Zanella, 2011a] G. Lamperti and M. Zanella. Context-sensitive diagnosis of discrete-event systems. In T. Walsh, editor, *Twenty-Second International Joint Conference on Artificial Intelligence IJCAI'11*, volume 2, pages 969–975, Barcelona, E, 2011. AAAI Press.

[Lamperti and Zanella, 2011b] G. Lamperti and M. Zanella. Monitoring of active systems with stratified uncertain observations. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 41(2):356–369, 2011.

[Paoli and Lafortune, 2008] A. Paoli and S. Lafortune. Diagnosability analysis of a class of hierarchical state machines. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 18(3):385–413, 2008.

[Pencolé and Cordier, 2005] Y. Pencolé and M.O. Cordier. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170, 2005.

[Pencolé *et al.*, 2001] Y. Pencolé, M.O. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *Twelfth International Workshop on Principles of Diagnosis DX'01*, pages 151–158, San Sicario, I, 2001.

[Pencolé, 2000] Y. Pencolé. Decentralized diagnoser approach: application to telecommunication networks. In *Eleventh International Workshop on Principles of Diagnosis DX'00*, pages 185–192, Morelia, MX, 2000.

[Qiu and Kumar, 2006] W. Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 36(2):384–395, 2006.

[Sampath *et al.*, 1996] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[Zhao and Ouyang, 2008] X. Zhao and D. Ouyang. Model-based diagnosis of discrete event systems with an incomplete system model. In *Eigtheenth European Conference on Artificial Intelligence ECAI'2008*, pages 189–193. IOS Press, Amsterdam, NL, 2008.

[Zhao *et al.*, 2012] X. Zhao, D. Ouyang, L. Zhang, X. Wang, and Y. Mo. Reasoning on partially-ordered observations in online diagnosis of DESs. *AI Communications*, 25(4):285–294, 2012.

A Spectrum of Diagnosis Approaches

Alban Grastien[1,2]
[1]Optimisation Research Group, NICTA, Australia
[1]Artificial Intelligence Group, Australian National University, Australia
e-mail: first.last@nicta.com.au

Abstract

This article argues that, regardless of the modeling framework, most diagnosis approaches can be classified in three categories or a combination of those categories. The purpose of this article is therefore to improve the understanding of diagnostic techniques used across the different modeling frameworks.

1 Introduction

Several aspects of diagnosis have been studied by the research community: problem modeling, i.e., how to represent the system, the observations, and the problem that is to be solved; problem solving, i.e., how to find the solution to the diagnosis problem; and diagnosis properties such as diagnosability, minimisation of sensors, etc. This paper concentrates on the second issue.

Much work has been dedicated to solving diagnosis problems because it is hard in general. In many situations, naive, brute-force, approaches are simply not practical, and even sophisticated methods fail to scale to real-world problems.

This paper presents a classification of diagnosis approaches in three categories: system state tracking, fault mode characterisations, and consistency checking. The objective here is to draw a coherent picture of the approaches that have been developed independently in different sub-communities primarily defined by the system modeling framework they rely on. For instance diagnosis of discrete event systems has long ignored the 'consistency checking' approach mostly for historical reasons (the seminal work been developed by Control theorists).

The document is divided as follows. Section 2 gives a generic definition of diagnosis. Section 3 presents the three approaches to diagnosis. Section 4 digresses on how diagnosability can influence the diagnostic approach. Finally Section 5 illustrates how algorithms have been developed for the three approaches, regardless of the modeling framework.

2 A Diagnosis Definition

The definition of diagnosis presented here and used later in the paper is very generic and practical implementations may be quite different from the definitions.

Model-based diagnosis uses a *model* that represents the possible set of system behaviours. Let us write $\mathbb{B}$ the set of behaviours allowed by the model and $\beta \in \mathbb{B}$ a single behaviour. The model also indicates what observations a given behaviour will produce. The *observation* (assumed deterministic) of behaviour β is $\sigma = obs(\beta)$ and the set of possible observations is $\mathbb{O} \ni \sigma$. Finally, diagnosis is about detecting and identifying the fault mode that the system is running under. The *fault mode* of behaviour β is $\delta = mode(\beta)$ and the set of modes is $\mathbb{M} \ni \delta$, which is generally the power set of a set of faults.

A diagnosis problem is defined as a model and an observation. The (consistency-based) diagnosis is defined as the set of modes that are logically consistent with the model and the observation, or a proper subset of this set, i.e., the set of modes exemplified by a behaviour consistent with the observation.

$$\Delta(\langle \mathbb{B}, obs, mode, \sigma \rangle) = \{\delta \in \mathbb{M} \mid \exists \beta \in \mathbb{B}.\ obs(\beta) = \sigma \wedge mode(\beta) = \delta\}. \quad (1)$$

3 Three Approaches

3.1 System State Tracking

The first approach to solve diagnosis is to have a literal interpretation of Equation 1. In this approach, the diagnosis is computed in two successive phases:

1. The set of system behaviours consistent with the observations is computed. We should call these behaviours *explanations* (although they should not be mistaken with the homonymous term from abductive diagnosis):
$$Expl = \{\beta \in \mathbb{B} \mid obs(\beta) = \sigma\}.$$
2. The diagnosis is then extracted from the set of explanations. It is generally assumed that this task is rather easy since it simply consists in ignoring the non-diagnosis-related information (internal variables) of the explanations:
$$\Delta = \{\delta \in \mathbb{M} \mid \exists \beta \in \mathbb{B}.\ mode(\beta) = \delta\}.$$

One clear advantage of this approach is that it is quite interactive. Because the set of explanations is computed, it is possible to request more details on-the-fly (for instance, return a full behaviour of the system that supports the diagnosis) or to refine the diagnosis by adding more information (a typical application

being the sequential diagnosis or the incremental diagnosis for dynamic systems). Practical implementations often keep only the relevant part of the explanations: for dynamic systems, only the final belief state.

The main issue of this approach is that it requires some heavy on-line processing. Computing *Expl* can be quite demanding and even the second phase can be hard (in particular if *Expl* is exponentially large compared to the original model). Most of the effort in this approach has therefore been spent on finding representations of *Expl* that i) are compact and that ii) allow for easy extraction of the diagnostic information. This approach also sometimes requires a strong-fault model, i.e., a model that can predict the behaviour of the system in faulty situations. Another issue is that it can be hard to distribute the computation (for instance, using multi-core processors).

3.2 Fault Mode Characterisation

The second approach to solve diagnosis consists in computing and using distinctive features of observations of fault modes. For instance, certain alarms may be explicit enough that the diagnosis can be derived without finding a precise explanation that supports this diagnosis; a perhaps extreme example would be an 100% reliable alarm "component A broken", which would be sufficient to produce the diagnosis "component A broken" without need to explain all other observations.

One way to formalise this approach would be to define a set of pair $\langle \mathbb{O}_i, \mathbb{M}_i \rangle$ with the following semantics:

$$\sigma \in \mathbb{O}_i \Rightarrow \Delta \cap \mathbb{M}_i = \emptyset.$$

For instance, if two sensors are supposed to return the same value in nominal condition, then $\mathbb{O}_i$ would represent the set of observations where the two sensors return different values and $\mathbb{M}_i$ would be (a subset of) the set that does not contain the nominal mode. It is assumed that $\mathbb{O}_i$ can be represented compactly and the inclusion $\sigma \in \mathbb{O}_i$ can be tested efficiently. Notice that the model $\mathbb{B}$ is missing in this definition, which does not mean that the approach is not model-based since i) the characterisations can be generated from a model and ii) the characterisations can be seen as models.

This approach is very appealing because it is generally quite efficient (most of the work is done off-line). Furthermore the characterisations are often readable and can be validated by an expert or even hand-written by an expert.

There are several issues with this approach. First, the number of characterisations is often very large, for instance if a unique characterisation is defined for each fault mode, and the number of fault mode is exponential (power set of a set of faults). Worst though, there are often many characterisations per fault mode (or per fault), especially in highly reconfigurable systems or systems that can be used, or react differently, in a wide range of situations.

A second line of research for this approach is the automatic generation of the characterisations.

On a side note, state-based observations can be seen as particular cases of characterisation (each state is characterised by what observations can be made in this state). Such characterisation can be trivial (problem inputs [Cordier and Largouët, 2001]) or not [Blackhall and Kan John, 2008; Daigle *et al.*, 2010; Bayoudh *et al.*, 2009].

3.3 Consistency Checking

The third approach to diagnosis is based on consistency checks. Each check is defined as a set of assumptions on the system behaviour. An oracle (for instance, a theorem prover) verifies whether these assumptions are consistent with the model and the observations. The checks are generated dynamically and the diagnosis is deduced from the result of the checks.

Formally, a check is a subset of system modes $\mathbb{M}_i \subseteq \mathbb{M}$ and the result of the check is i) either a mode $\delta \in \mathbb{M}_i \cap \Delta$ if one such mode exists or ii) a superset $\mathbb{M}'_i \supseteq \mathbb{M}_i$ such that $\mathbb{M}'_i \cap \Delta = \emptyset$ ($\mathbb{M}'_i$ is called a *conflict*; in the worst-case $\mathbb{M}'_i = \mathbb{M}_i$).

This approach is very appealing in that it can adapt quite efficiently to the problem at hand. For instance, assume that i) the system behaviour is nominal and ii) we are looking only for the minimal diagnosis. Then the diagnosis can be found with only one consistency check. The other two approaches on the other hand could require quite expensive (pre-)computation.

There are several issues with this approach however: it is unable to generate all the diagnoses in general, as this would require an exponential number of checks; it is often limited to (possibly all) minimal diagnoses. Another problem is that the computation time is very unpredictable in general. An issue that is being addressed is the solving of the checks, especially for dynamic systems (reduces to model-checking).

3.4 Discussion

The three categories of algorithms provided above bear some conceptual similarities but behave quite differently. State tracking and fault mode characterisation can be similar in that the characterisations are often computed by computing all possible results a state tracking could provide (typically, the Sampath diagnoser for DES [Sampath *et al.*, 1995]). On the other hand, a characterisation can be computed by asserting assumptions (as in consistency checking) and deriving the logical consequences.

Fault mode characterisation relies on significant off-line preprocessing but expects little on-line operations (which is why it has been pushed by the Control community). On the other hand, system tracking and consistency checking require an important computation at runtime but expect that the specificity of the diagnostic problem, i.e., the observations of the behaviour to explain, will allow for swift resolution.

In this sense, any off-line preprocessing aimed at altering the model (for instance, model abstractions) to simplify the diagnosis problem has a fault mode characterisation flavor.

4 The Role of Diagnosability

Diagnosability is the property that the faults can be identified when they occur on the system. This property makes the system easier to control and efforts should be made at design stage to ensure diagnosability.

There is another benefit from diagnosability that was recognised in the last decade, which is that diagnosability can make the diagnosis problem easier to solve.

Essentially, diagnosability tells us that there is a (precise) solution to the problem, and it is therefore worth searching of it.

Rather than performing diagnosis with a set of fault modes $\mathbb{M}$, one might actually want to solve k independent diagnosis subproblems with different fault modes $\mathbb{M}_i$ such that $\mathbb{M} \subseteq \mathbb{M}_1 \times \ldots \times \mathbb{M}_k$. The diagnosis is then the set of modes accepted by each subdiagnosis:

$$\Delta = \mathbb{M} \cap (\Delta_i \times \ldots \times \Delta_k) .$$

This decomposition improves performance if the time necessary to solve the diagnosis problem with $\mathbb{M}$ is bigger than that of solving all the subproblems. This generally requires i) that $|\mathbb{M}| \gg |\mathbb{M}_1| + \ldots + |\mathbb{M}_k|$ and ii) that abstractions can be applied for each subproblem.

The potential issue here is a decrease in precision. Consider a diagnosis with two faults a and b and four fault modes: $\mathbb{M} = \{\{\neg a, \neg b\}, \{a, \neg b\}, \{\neg a, b\}, \{a, b\}\}$. This problem can be decomposed in two subproblems each dedicated to one specific fault: the first subproblem has modes $\{\neg a\}$, $\{a\}$ and the second one has modes $\{\neg b\}$, $\{b\}$. If the diagnoses are $\{\neg a, b\}$ and $\{a, \neg b\}$ (exactly one fault occurred), then the diagnoses will be $\{\neg a\}$, $\{a\}$ for the first subproblem and $\{\neg b\}$, $\{b\}$ for the second one, which will translate to $\Delta = \mathbb{M}$, hence the precision loss.

It has been recognised that, if each fault is diagnosable, then the decomposition presented above is precise [Cordier *et al.*, 2006], and this result has been extended to other decompositions [Grastien and Torta, 2011].

Decompositions as presented above have been widely used. Characterisation methods can be seen as implementations of such decompositions: the set $\mathbb{M}_i$ presented in Subsection 3.2 bears many similarities with decompositions. The abstraction is implicit in this type of approach since the characterisations will generally refer to a specific subset of the sensors.

However characterisations-based approaches are not the only ones that can benefit from decomposition. Since each subproblem can be solved in parallel, tracking-based approaches and consistency-based approaches can benefit from it (consistency-based approaches can be seen as a dynamic implementation of this scheme with binary sets $\mathbb{M}_i$). Notice however that when abstraction is applied (which is one reason to perform decomposition, at least for the tracking approaches), this can reduce the ability to produce global explanations that support a diagnosis, since the diagnosis is not computed globally.

Diagnosability can also be used to parallelise computation in consistency checking approaches. For instance, if fault f is diagnosable, one can start two searches in parallel, one looking for an explanation including fault f and the other one excluding f; the solution will be found as soon as one search terminates (either by finding an explanation, or by proving that no such explanation exists).

5 Approaches within Modeling Frameworks

The purpose of this section is to illustrate how those different algorithms have been implemented in different modeling frameworks.

5.1 Circuits

The Theory of diagnosis from first principles from Reiter [1987] is an implementation of the consistency checking approach. Starting from the best diagnosis (no fault), consistency of a fault mode is tested and certified or nullified together with a larger set of modes through a conflict. Under the assumption that the model is weakly-faulty (the effect of the faults on the model is not modeled), this algorithm even returns all the diagnoses.

More recently, SAFARI [Feldman *et al.*, 2010] computes some diagnoses and tries to improve them by removing faults that are not essential in the current mode. Reduction to SAT has been proposed [Smith *et al.*, 2005] and recently made scalable [Metodi *et al.*, 2012]. Those algorithms are based on consistency-checking.

The approach from De Kleer and Williams [1987] is very interesting in that it does not quite fall in any of the approaches presented in this paper. It consists in using an ATMS to derive facts from any assumption on the faults. Conflicts are then generated, which allow to compute the diagnoses (the hitting sets of the conflicts). It sits somewhere between state tracking and consistency checking: it derives all the facts it can (which is similar to the tracking approach) but it also dynamically generates conflicts before producing the diagnosis.

The work initiated by Provan and Darwiche sits between the tracking zone and the mode characterisation: on the one hand, the goal is to find a compact representation of the system belief state, on the other hand, a great deal of energy is performed offline to build a structure that will allow this on-line efficiency, including by applying abstraction [Provan, 2001]. The model, the observations, and the fault modes are all propositional formula, and the diagnosis is computed by computing the conjunction of the model and the observations and projecting the result on the mode variables [Huang and Darwiche, 1996; Darwiche, 1998]. Because such an algorithm is exponential in general, and often also in practice, efforts were made to identified classes of representations for propositional formula (improved BDDs [Torta and Torasso, 2004]) that would limit the constraints on the algorithm [Darwiche and Marquis, 2002; Huang and Darwiche, 2007; Siddiqi and Huang, 2011]. Substantial work was also dedicated to finding structural properties that can be used to speed up the computation of diagnosis [Huang and Darwiche, 1996; Darwiche, 1998; Siddiqi and Huang, 2007; Siddiqi, 2011].

5.2 Discrete Event Systems

Discrete event systems (DES) are dynamic systems whose evolution is modeled by a finite set of events.

The first works in (model-based) diagnosis of DES have used characterisation and are the chronicles and the Sampath diagnoser.

A chronicle is a collection of observable events that are temporally constraints [Cordier and Dousson, 2000]. If this collection of events is recognised in the flow of observations generated by the system, what information associated with the chronicle (generally a fault) is identified. The work in this topic

has concentrated on making the recognition fast [Dousson and Le Maigat, 2007] and on generating the chronicles automatically [Guerraz and Dousson, 2004; Pencolé and Subias, 2009]

The Sampath diagnoser is an extreme version of characterisation [Sampath *et al.*, 1995]. For each fault mode δ, the set $\mathbb{O}_\delta$ of the observations consistent with this mode is generated: $\mathbb{O}_\delta = \{\sigma \in \mathbb{O} \mid \exists \beta \in \mathbb{B}.\ \sigma = obs(\beta) \wedge \delta = mode(\beta)\}$. These sets are languages that can be represented by DFAs (deterministic finite automata, assuming the state space of the DES was finite) where each state is labeled with the diagnosis. Given a sequence of observations, the diagnosis algorithm only needs to follow the single path on the diagnoser associated with these observations, and extract the diagnosis at the final state. The main problem of the diagnoser is its size: exponential in the number of state in the DES and in the number of fault modes, i.e., generally double exponential in the number of state variables and in the number of faults, which makes it impractical but for tiny problems [Rintanen, 2007]. There has been some work on reducing the size of the diagnoser using abstraction (bisimulation) [Marchand and Rozé, 2002], using BDDs [Schumann *et al.*, 2004], or by decomposing the fault modes [Pencolé *et al.*, 2006]. In this last paper, the decomposition is based on building a diagnoser for each fault; there could be potential for research here in building several diagnosers on different types of decomposition (cf. [Grastien and Torta, 2011]).

The Artificial Intelligence community has mostly worked on the tracking approach [Zanella and Lamperti, 2003]. Because this approach is very expensive in general, many algorithms have been used to exploit the symmetries in the model, e.g., decentralised/distributed techniques [Pencolé and Cordier, 2005; Su and Wonham, 2005; Cordier and Grastien, 2007; Kan John and Grastien, 2008; Kan John *et al.*, 2010], symbolic tools (BDDs) [Schumann *et al.*, 2007], Petri nets [Aghasaryan *et al.*, 1998; Jiroveanu and Boël, 2005; Cabasino *et al.*, 2010].

The work on the consistency-based approach has mostly concentrated on the implementation of the consistency checker, either a planner [McIlraith, 1998; Sohrabi *et al.*, 2010; Haslum and Grastien, 2011], a model-checker [Cordier and Largouët, 2001], a SAT solver [Grastien *et al.*, 2007] or an integer linear programming solver [Basile *et al.*, 2009; Dotoli *et al.*, 2009]. The second level of the approach, i.e., what consistency checks should be generated is a relatively novel topic [Grastien *et al.*, 2011; 2012].

There has been some attempts at combining different approaches. Lamperti and Zanella compute the diagnosis with the state-tracking approach, but reuse the computed diagnosis to build characterisations [Lamperti and Zanella, 2004].

It should be possible to combine characterisations with other approaches: for instance, use characterisation to quickly find a diagnosis, and use more expensive approaches were this one to fail. Similarly consistency-based approaches could implement decentralised approaches used in state tracking, by trying to prove/disprove the occurrence of a fault locally, before considering more system-wide model.

5.3 Continuous and Hybrid Systems

One obvious way to diagnose continuous and hybrid systems is to discretise them and use techniques from diagnosis of DES [Bresolin and Capiluppi, 2011]

Characterisation approaches are often used for diagnosis of continuous and hybrid systems, possibly because they run very fast, once the characterisations have been found. These characterisations are known as possible conflicts [Pulido and Alonso González, 2004], analytical redundancy relations [Staroswiecki and Comtet-Varga, 2001], minimal structurally overdetermined sets [Krysander *et al.*, 2010], etc.

In essence, these *indicators* are built by infering the state of internal variables from some observations and predicting other observations from these inferences. However certain functions are difficult to invert and work has been devoted to generate these indicators nevertheless [de Flaugergues *et al.*, 2010]. The second major issue is coping with the number of indicators by generating them on-the-fly [Heintz *et al.*, 2008; Bayoudh *et al.*, 2009].

The tracking approach has also been used extensively. Multi-mode Kalman filters compute the probability distribution on the state space [Blom and Bar-Shalom, 1988] and most of the work has been dedicated to finding good approximations of these distributions [Benazera and Travé-Massuyès, 2009]. Particle filters [Arulampalam *et al.*, 2002] samples the probability distribution to keep the representation of this distribution small and the simulation easy. Similarly, HyDE [Narasimhan and Brownston, 2007] maintains a list of candidates that are discarded when their weight goes below a threshold.

The consistency checking approach has been proposed by Dague [1994] but seems to have been abandonned until recently. Ernits & Dearden [2011] use an SMT solver to solve the diagnosis checks. Both approaches are done on the steady state.

6 Conclusion

This article argued that diagnosis algorithms can be classified in three categories: tracking the state of the system, characterising the different system modes, and performing consistency checks between the model, the observations, and some assumptions.

Interestingly, the same approaches have been developed regardless of the modeling framework. Researchers working on different types of model but with the same approach to diagnose have been faced with the same issues.

Acknowledgments

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

[Aghasaryan *et al.*, 1998] A. Aghasaryan, É. Fabre, A. Benveniste, R. Boubour, and C. Jard. Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri Nets. *Journal of Discrete Event Dynamical Systems (JDEDS)*, 8(2):203–231, 1998.

[Arulampalam *et al.*, 2002] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing (TSP)*, 50(2):174–188, 2002.

[Basile *et al.*, 2009] F. Basile, P. Chiacchio, and G. De Tommasi. An efficient approach for online diagnosis of discrete event systems. *IEEE Transactions on Automatic Control (TAC)*, 54(4):748–759, 2009.

[Bayoudh *et al.*, 2009] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Diagnosis of a class of non linear hybrid systems by on-line instantiation of parameterized analytic redundancy relations. In *20th International Workshop on Principles of Diagnosis (DX-09)*, pages 283–290, 2009.

[Benazera and Travé-Massuyès, 2009] E. Benazera and L. Travé-Massuyès. Set-theoretic estimation of hybrid system configurations. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 39(5):1277–1290, 2009.

[Blackhall and Kan John, 2008] L. Blackhall and P. Kan John. Model-based diagnosis of hybrid dynamical networks for fault tolerant control. In *19th International Workshop on Principles of Diagnosis (DX-08)*, pages 239–246, 2008.

[Blom and Bar-Shalom, 1988] H. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control (TAC)*, 33(8):780–783, 1988.

[Bresolin and Capiluppi, 2011] D. Bresolin and M. Capiluppi. A game-theoretic approach to fault diagnosis of hybrid systems. In *Second International Symposium on Games, Automata, Logics and Formal Verification (GandALF-11)*, pages 237–249, 2011.

[Cabasino *et al.*, 2010] M. P. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, 2010.

[Cordier and Dousson, 2000] M.-O. Cordier and Ch. Dousson. Alarm driven monitoring based on chronicles. In *Fourth IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SafeProcess-00)*, pages 286–291, 2000.

[Cordier and Grastien, 2007] M.-O. Cordier and A. Grastien. Exploiting independence in a decentralised and incremental approach of diagnosis. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[Cordier and Largouët, 2001] M.-O. Cordier and Ch. Largouët. Using model-checking techniques for diagnosing discrete-event systems. In *12th International Workshop on Principles of Diagnosis (DX-01)*, pages 39–46, 2001.

[Cordier *et al.*, 2006] M.-O. Cordier, L. Travé-Massuyès, and X. Pucel. Comparing diagnosability in continuous and discrete-event systems. In *17th International Workshop on Principles of Diagnosis (DX-06)*, pages 55–60, 2006.

[Dague, 1994] Ph. Dague. Model-based diagnosis of analog electronic circuits. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 11:439–492, 1994.

[Daigle *et al.*, 2010] M. Daigle, I. Roychoudhury, G. Biswas, and X. Koutsoukos. An event-based approach to distributed diagnosis of continuous systems. In *21st International Workshop on Principles of Diagnosis (DX-10)*, pages 15–22, 2010.

[Darwiche and Marquis, 2002] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[Darwiche, 1998] A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research (JAIR)*, 8:165–222, 1998.

[de Flaugergues *et al.*, 2010] V. de Flaugergues, V. Cocquempot, M. Bayart, and M. Pengov. On non-invertibilities for structural analysis. In *21st International Workshop on Principles of Diagnosis (DX-10)*, pages 23–30, 2010.

[de Kleer and Williams, 1987] J. de Kleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence (AIJ)*, 32:97–130, 1987.

[Dotoli *et al.*, 2009] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich. On-line detection in discrete event systems by Petri nets and integer linear programming. *Automatica*, 45:2665–2672, 2009.

[Dousson and Le Maigat, 2007] Chr. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 324–329, 2007.

[Ernits and Dearden, 2011] J. Ernits and R. Dearden. Towards diagnosis modulo theories. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 249–256, 2011.

[Feldman *et al.*, 2010] A. Feldman, G. Provan, and A. van Gemund. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research (JAIR)*, 38:371–413, 2010.

[Grastien and Torta, 2011] A. Grastien and G. Torta. Reformulation for the diagnosis of discrete-event systems. In *Ninth Symposium on Abstraction, Reformulation and Approximation (SARA-11)*, pages 42–49, 2011.

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, pages 305–310, 2007.

[Grastien *et al.*, 2011] A. Grastien, P. Haslum, and S. Thiébaux. Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 60–67, 2011.

[Grastien *et al.*, 2012] A. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: theory and practice. In *13th International Conference on the Principles of Knowledge Representation and Reasoning (KR-12)*, 2012.

[Guerraz and Dousson, 2004] B. Guerraz and Ch. Dousson. Chronicles construction starting from the fault model of the system to diagnose. In *15th International Workshop on Principles of Diagnosis (DX-04)*, pages 51–56, 2004.

[Haslum and Grastien, 2011] P. Haslum and A. Grastien. Diagnosis as planning: two case studies. In *Fifth Scheduling and Planning Applications Workshop (SPARK-11)*, pages 37–44, 2011.

[Heintz *et al.*, 2008] F. Heintz, M. Krysander, J. Roll, and E. Frisk. FlexDx: a reconfigurable diagnosis framework. In *19th International Workshop on Principles of Diagnosis (DX-08)*, pages 79–86, 2008.

[Huang and Darwiche, 1996] C. Huang and A. Darwiche. Inference in belief networks: a procedural guide. *International Journal of Approximate Reasoning (IJAR)*, 15(3):225–263, 1996.

[Huang and Darwiche, 2007] J. Huang and A. Darwiche. The language of search. *Journal of Artificial Intelligence Research (JAIR)*, 29:191–219, 2007.

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

[Kan John and Grastien, 2008] P. Kan John and A. Grastien. Local consistency and junction tree for diagnosis of discrete-event systems. In *18th European Conference on Artificial Intelligence (ECAI-08)*, 2008.

[Kan John *et al.*, 2010] P. Kan John, A. Grastien, and Y. Pencolé. Synthesis of a distributed and accurate diagnoser. In *21st International Workshop on Principles of Diagnosis (DX-10)*, pages 209–216, 2010.

[Krysander *et al.*, 2010] M. Krysander, J. Åslund, and E. Frisk. A structural algorithm for finding testable submodels and multiple fault isolability analysis. In *21st International Workshop on Principles of Diagnosis (DX-10)*, pages 79–86, 2010.

[Lamperti and Zanella, 2004] G. Lamperti and M. Zanella. A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 34(5):2222–2244, 2004.

[Marchand and Rozé, 2002] H. Marchand and L. Rozé. Diagnostic de pannes sur des systèmes à événements discrets : une approche à base de modèles symboliques. In *13th Congrès francophone AFRIF-AFIA (RFIA-02)*, pages 191–200, 2002.

[McIlraith, 1998] Sh. McIlraith. Explanatory diagnosis: conjecturing actions to explain observations. In *Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 167–177, 1998.

[Metodi *et al.*, 2012] A. Metodi, R. Stern, M. Kalech, and M. Codish. Compiling model-based diagnosis to Boolean satisfaction. In *26th Conference on Artificial Intelligence (AAAI-12)*, 2012.

[Narasimhan and Brownston, 2007] S. Narasimhan and L. Brownston. HyDE - a general framework for stochastic and hybrid model-based diagnosis. In *18th International Workshop on Principles of Diagnosis (DX-07)*, pages 162–169, 2007.

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

[Pencolé and Subias, 2009] Y. Pencolé and A. Subias. A chronicle-based diagnosability approach for discrete timed-event systems: application to web-services. *Journal of Universal Computer Science (jucs)*, 15(17):3246–3272, 2009.

[Pencolé *et al.*, 2006] Y. Pencolé, D. Kamenetsky, and A. Schumann. Towards low-cost fault diagnosis in large component-based systems. In *Sixth IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SafeProcess-06)*, 2006.

[Provan, 2001] G. Provan. Hierarchical model-based diagnosis. In *12th International Workshop on Principles of Diagnosis (DX-01)*, pages 167–174, 2001.

[Pulido and Alonso González, 2004] B. Pulido and C. Alonso González. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 34(5):2192–2206, 2004.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)*, 32(1):57–95, 1987.

[Rintanen, 2007] J. Rintanen. Diagnosers and diagnosability of succinct transition systems. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 538–544, 2007.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 40(9):1555–1575, 1995.

[Schumann *et al.*, 2004] A. Schumann, Y. Pencolé, and S. Thiébaux. Symbolic models for diagnosing discrete-event systems. In *16th European Conference on Artificial Intelligence (ECAI-04)*, pages 1085–1086, 2004.

[Schumann *et al.*, 2007] A. Schumann, Y. Pencolé, and S. Thiébaux. A spectrum of symbolic on-line diagnosis approaches. In *22nd Conference on Artificial Intelligence (AAAI-07)*, 2007.

[Siddiqi and Huang, 2007] S. Siddiqi and J. Huang. Hierarchical diagnosis of multiple faults. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 581–586, 2007.

[Siddiqi and Huang, 2011] S. Siddiqi and J. Huang. Sequential diagnosis by abstraction. *Journal of Artificial Intelligence Research (JAIR)*, 41:329–365, 2011.

[Siddiqi, 2011] S. Siddiqi. Computing minimum-cardinality diagnoses by model relaxation. In *22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011.

[Smith *et al.*, 2005] A. Smith, A. Veneris, M. F. Ali, and A. Viglas. Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 24(10):1606–1621, 2005.

[Sohrabi *et al.*, 2010] Sh. Sohrabi, J. Baier, and Sh. McIlraith. Diagnosis as planning revisited. In *12th International Conference on the Principles of Knowledge Representation and Reasoning (KR-10)*, pages 26–36, 2010.

[Staroswiecki and Comtet-Varga, 2001] M. Staroswiecki and G. Comtet-Varga. Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems. *Automatica*, 37:687–699, 2001.

[Su and Wonham, 2005] R. Su and W. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 50(12):1923–1935, 2005.

[Torta and Torasso, 2004] G. Torta and P. Torasso. The role of OBDDs in controlling the complexity of model based diagnosis. In *15th International Workshop on Principles of Diagnosis (DX-04)*, pages 9–14, 2004.

[Zanella and Lamperti, 2003] M. Zanella and G. Lamperti. *Diagnosis of active systems*. Kluwer Academic Publishers, 2003.

Sequential Decision Process Supported by a Compositional Model*

Radim Jiroušek
University of Economy, Prague, Faculty of Management, Jindřichuv Hradec
and
Inst. Information Theory and Automation, Acad. of Sciences of the Czech Republic
e-mail: radim@utia.cas.cz

Abstract

The goal of the paper is to describe a classical sequential decision process that is often used for both medical and technical diagnosis making. Though we do not perform a detailed analysis of its computational complexity, we show that the whole process can be realized for probability distributions of very high dimensionality under the assumption that the distribution is represented as a compositional model of special properties.

1 Introduction

The basic idea, upon which a sequential decision process is based, is simple. The goal is to find out the evidence supporting a decision with a required certainty. To express it more precisely let us use the language of probability theory. Let X_δ denote a decision variable whose values correspond to individual decisions (e.g., diagnoses). If κ is a probability distribution describing a relationship among the feature variables and the decision variable, we want to find out a subset E of feature variables whose values $e \in \mathbb{X}_E$ (evidence) observed in the situation in question yields

$$\kappa(X_\delta = d|E = e) \geq \varepsilon$$

for some $d \in \mathbb{X}_\delta$. In a sequential process, the set E is gradually constructed by adding one variable at each step. Naturally, we want to keep the set E as small as possible, or generally, if some weights (or costs) connected with individual variables are given, we want to get the cheapest possible decision process.

Quite often, sequential decision processes are represented in a form of a decision tree. However, in many practical problems a fixed tree does not suit the situations. For example, in case of a medical diagnosis making, a patient visits a physician with some complains, i.e., with symptoms (values of variables) that need not be at the beginning of the decision tree, and still they should be included in the set E. Similarly, faults of a technical device manifest at the very beginning in different ways (or, it may be observed by different users in different ways). Not speaking about a newly appointed manager whose task is to strengthen a company. To diagnose weaknesses of the enterprize she always starts with different prior knowledge. Moreover, costs assigned to variables may differ from case to case. A patient may come to a specialist with the results of some expensive laboratory tests that were already performed by another physician, so the specialist has it free. This is why we prefer not to compute a single decision tree in advance but to construct a sequential decision process based on a multidimensional probability distribution at time of its application.

When speaking about dimensionality of probability distributions, it is clear that in practical situations we have to have in mind rather thousands than tens of variables. Therefore we have to have a tool enabling us to represent and compute with such large distributions.

For this purpose, Bayesian networks [Jensen and Nielsen, 2007] that were developed in 1980s, or other graphical models [Lauritzen, 1996] are often used. In contrast to this, in this paper we want to enhance application of another, non-graphical approach, so called compositional models that were proposed for multidimensional probability distributions in [Jiroušek, 1997].

The paper is organized as follows: Section 2 introduces a necessary notation and brings a brief introduction to compositional models (more on this topic can be found in [Jiroušek, 2011]). In Section 3 we show that it is possible to compute conditionals even for probability distributions of very high dimensions when the distributions are represented in a form of compositional models of special properties. Section 4 is devoted to the description of the sequential decision process.

2 Compositional Model

In this text we use a simplified version of the notation from [Jiroušek, 2011]. We deal with a finite system of finite-valued random variables $N = \{X_1, X_2, \ldots, X_n\}$. A set of values of variable X_i, which is denoted $\mathbb{X}_i$, is assumed to have at least two elements. The set of all combinations of the considered values is denoted $\mathbb{X}_N = \mathbb{X}_1 \times \mathbb{X}_2 \times \ldots \times \mathbb{X}_n$. Analogously, for $K \subset N$, $\mathbb{X}_K = \times_{i \in K} \mathbb{X}_i$. As indicated in Introduction, we consider one variable $X_\delta \in N$ to be a decision variable.

Distributions of the considered variables are denoted by Greek letters (π, κ, ν, μ) with possible indices; thus for $K \subseteq N$, we can consider a distribution $\pi(K)$, which is a $|K|$-dimensional distribution and $\pi(x)$ denotes a value of probability distribution π for point $x \in \mathbb{X}_K$.

For a probability distribution $\pi(K)$ and $J \subset K$, we often consider a *marginal distribution* $\pi^{\downarrow J}$ of π, which can be

*The research was partially supported by the grant of GA ČR no. 403/12/2175.

computed for all $x \in \mathbb{X}_J$ by

$$\pi^{\downarrow J}(x) = \sum_{y \in \mathbb{X}_K : y^{\downarrow J} = x} \pi(y),$$

where $y^{\downarrow J}$ denotes the *projection* of $y \in \mathbb{X}_K$ into $\mathbb{X}_J$. For computation of marginal distributions we need not exclude situations when $J = \emptyset$. By definition, we get $\pi^{\downarrow \emptyset} = 1$.

Having two distributions $\pi(K)$ and $\kappa(K)$, we say that κ dominates π (in symbol $\pi \ll \kappa$) if for all $x \in \mathbb{X}_K$

$$\kappa(x) = 0 \implies \pi(x) = 0$$

(notice that some authors say that π is absolutely continuous with respect to κ; however the latter notion is used mainly when considering continuous spaces).

One of the most important notions supporting an efficient representation of multidimensional probability distributions is a famous concept of conditional independence (se e.g. [Lauritzen, 1996] or [Studený, 2005]).

For a probability distribution $\pi(K)$ and three disjoint subsets $L, M, R \subseteq K$ such that both $L, M \neq \emptyset$, we say that groups of variables L and M are *conditionally independent* given R (in symbol $L \perp\!\!\!\perp M | R\ [\pi]$) if

$$\pi^{\downarrow L \cup M \cup R} \pi^{\downarrow R} = \pi^{\downarrow L \cup R} \pi^{\downarrow M \cup R}.$$

2.1 Operator of Composition

For the compositional models, the key notion is that of a composition.

Definition 1. *For two arbitrary distributions $\pi(K)$ and $\kappa(L)$, for which $\pi^{\downarrow K \cap L} \ll \kappa^{\downarrow K \cap L}$, their* composition *is, for each $x \in \mathbb{X}_{(L \cup K)}$, given by the following formula*

$$(\pi \triangleright \kappa)(x) = \frac{\pi(x^{\downarrow K}) \kappa(x^{\downarrow L})}{\kappa^{\downarrow K \cap L}(x^{\downarrow K \cap L})}.$$

In case $\pi^{\downarrow K \cap L} \not\ll \kappa^{\downarrow K \cap L}$, the composition remains undefined.

The following simple assertion answers the question: what is the result of composition of two distributions? For its proof the reader is referred to [Jiroušek, 2011].

Lemma 1. *Let $\pi(K), \kappa(L)$ be probability distributions such that $\pi^{\downarrow L \cap K} \ll \kappa^{\downarrow L \cap K}$. Then $\pi \triangleright \kappa$ is a probability distribution of variables $(L \cup K)$ and its marginal distribution for variables K equals π:*

$$(\pi \triangleright \kappa)^{\downarrow K} = \pi.$$

The above presented assertion says that using the operator of composition one can easily construct a more-dimensional distribution from two low-dimensional ones. Naturally, it can be done only under the assumption of the respective dominance. On the other hand side, having a more-dimensional distribution one can, in some situations, factorize this distribution into two (or more) low-dimensional ones. This property is precisely expressed in the following assertion, the proof of which can also be found in [Jiroušek, 2011].

Lemma 2. *Consider a probability distribution $\pi(K)$, and three disjoint subsets $L, M, R \subseteq K$ such that both $L, M \neq \emptyset$. Then $L \perp\!\!\!\perp M | R\ [\pi]$ if and only if*

$$\pi^{\downarrow L \cup M \cup R} = \pi^{\downarrow L \cup R} \triangleright \pi^{\downarrow M \cup R}.$$

In [Jiroušek, 2011], many properties of the composition operator are proved. Those that are necessary in this paper are summarized in the following assertion.

Lemma 3. *Suppose $\pi(K)$, $\kappa(L)$ and $\mu(M)$ are such probability distributions that all the following expressions (compositions) are defined. Then the following statements hold.*

1. *In general, composition is not commutative:*
$$\pi \triangleright \kappa \neq \kappa \triangleright \pi.$$
2. *If π and κ are consistent, i.e., $\pi^{\downarrow K \cap L} = \kappa^{\downarrow K \cap L}$, then*
$$\pi \triangleright \kappa = \kappa \triangleright \pi.$$
3. *In general, composition is not associative:*
$$(\pi \triangleright \kappa) \triangleright \mu \neq \pi \triangleright (\kappa \triangleright \mu).$$
4. *If $L \supset (K \cap M)$. Then,*
$$(\pi \triangleright \kappa) \triangleright \mu = \pi \triangleright (\kappa \triangleright \mu).$$
5. *If $(K \cap L) \subseteq R \subseteq K \cup L$, then*
$$(\pi \triangleright \kappa)^{\downarrow R} = \pi^{\downarrow K \cap R} \triangleright \kappa^{\downarrow L \cap R}.$$
6. *If $K \supset (L \cap M)$, then*
$$(\pi \triangleright \kappa) \triangleright \mu = (\pi \triangleright \mu) \triangleright \kappa.$$

2.2 Generating Sequences

In the rest of the paper we will deal with sequences of low-dimensional probability distributions. To avoid some technical problems and the necessity of repeating some assumptions to excess, let us make the following three conventions.

First, whenever we speak about a distribution π_k, it will be a distribution $\pi_k(K_k)$. Thus, formula $\pi_1 \triangleright \ldots \triangleright \pi_m$, if it is defined, determines the distributions of variables $K_1 \cup \ldots \cup K_m$.

Since the operator of composition is not associative (see Lemma 3), the formulas like $\pi_1 \triangleright \ldots \triangleright \pi_m$ are, strictly speaking, ambiguous. Therefore, the second convention avoids this ambiguity saying that we always apply the operators from left to right. Thus

$$\pi_1 \triangleright \pi_2 \triangleright \pi_3 \triangleright \ldots \triangleright \pi_m = (((\pi_1 \triangleright \pi_2) \triangleright \pi_3) \triangleright \ldots \triangleright \pi_m),$$

and the parentheses will be used only when we want to change this default ordering. Therefore, to construct a multidimensional distribution it is sufficient to determine a sequence – we call it a *generating sequence* – of low-dimensional distributions.

The third forementioned convention is of a rather technical nature. Considering a generating sequence $\pi_1, \pi_2, \ldots, \pi_m$ we will assume that

$$\begin{aligned}
&\pi_1^{\downarrow K_2 \cap K_1} \ll \pi_2^{\downarrow K_2 \cap K_1},\\
&(\pi_1 \triangleright \pi_2)^{\downarrow K_3 \cap (K_1 \cup K_2)} \ll \pi_3^{\downarrow K_3 \cap (K_1 \cup K_2)},\\
&(\pi_1 \triangleright \pi_2 \triangleright \pi_3)^{\downarrow K_4 \cap (K_1 \cup K_2 \cup K_3)} \ll \pi_4^{\downarrow K_4 \cap (K_1 \cup K_2 \cup K_3)},\\
&\quad \vdots\\
&(\pi_1 \triangleright \ldots \triangleright \pi_{m-1})^{\downarrow K_m \cap (K_1 \cup \ldots \cup K_{m-1})}\\
&\qquad \ll \pi_m^{\downarrow K_m \cap (K_1 \cup \ldots \cup K_{m-1})},
\end{aligned}$$

i.e., that $\pi_1 \triangleright \pi_2 \triangleright \pi_3 \triangleright \ldots \triangleright \pi_m$ is defined and the result is a $|K_1 \cup \ldots \cup K_m|$-dimensional probability distribution.

However, not all generating sequences are equally efficient in their representations of multidimensional distributions. Among them, so-called perfect sequences hold an important position [Jiroušek, 1997].

Definition 2. *A generating sequence of probability distributions $\pi_1, \pi_2, \ldots, \pi_m$ is called* perfect *if*

$$\begin{aligned}
&\pi_1 \triangleright \pi_2 = \pi_2 \triangleright \pi_1,\\
&\pi_1 \triangleright \pi_2 \triangleright \pi_3 = \pi_3 \triangleright (\pi_1 \triangleright \pi_2),\\
&\quad\vdots\\
&\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m = \pi_m \triangleright (\pi_1 \triangleright \ldots \triangleright \pi_{m-1}).
\end{aligned}$$

From this definition, one can hardly see the importance of perfect sequences. This importance becomes clearer from the following characterization theorem (for the proofs of all the assertions presented in the rest of this section see [Jiroušek, 2011]).

Theorem 1. *A sequence of distributions $\pi_1, \pi_2, \ldots, \pi_m$ is perfect* iff *all the distributions from this sequence are marginals of the distribution $\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m$.*

Another advantageous property of the perfect sequences says that perfect sequences represent a unique distribution in the following sense.

Theorem 2. *If a sequence $\pi_1, \pi_2, \ldots, \pi_m$ and its permutation $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_m}$ are both perfect, then*

$$\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m = \pi_{i_1} \triangleright \pi_{i_2} \triangleright \ldots \triangleright \pi_{i_m}.$$

In the rest of the paper we will need two more facts expressed in the following assertions. The first says that any generating sequence can be transformed into a perfect sequence without influencing the resulting multidimensional distribution.

Theorem 3. *For any generating sequence $\pi_1, \pi_2, \ldots, \pi_m$, the sequence $\kappa_1, \kappa_2, \ldots, \kappa_m$ computed by the following process*

$$\begin{aligned}
&\kappa_1 = \pi_1,\\
&\kappa_2 = \kappa_1^{\downarrow K_2 \cap K_1} \triangleright \pi_2,\\
&\kappa_3 = (\kappa_1 \triangleright \kappa_2)^{\downarrow K_3 \cap (K_1 \cup K_2)} \triangleright \pi_3,\\
&\quad\vdots\\
&\kappa_m = (\kappa_1 \triangleright \ldots \triangleright \kappa_{m-1})^{\downarrow K_m \cap (K_1 \cup \ldots \cup K_{m-1})} \triangleright \pi_m
\end{aligned}$$

is perfect, and

$$\pi_1 \triangleright \ldots \triangleright \pi_m = \kappa_1 \triangleright \ldots \triangleright \kappa_m.$$

From the theoretical point of view, the process of perfectization described in Theorem 3 is simple. Unfortunately, it is not valid from the point of view of computational complexity. Namely, in the process marginals $(\kappa_1 \triangleright \ldots \triangleright \kappa_{r-1})^{\downarrow K_r \cap (K_1 \cup \ldots \cup K_{r-1})}$ must be computed. Unfortunately, this problem may be computationally very expensive [Jiroušek, 2000]. Therefore, in the next sections we will take advantage of a special generating sequences, namely those whose sequences of variables $K_1, K_2, \ldots, K_m$ meet the so called *running intersection property* (RIP):

$$\forall r = 2, \ldots, m \;\; \exists s (1 \leq s < r) \quad \left(K_r \cap (\bigcup_{k=1}^{r-1} K_k) \subseteq K_s \right).$$

Consider a perfectization process applied to such a RIP generating sequence $\pi_1, \pi_2, \ldots, \pi_m$. In this case, either $K_1 \supseteq K_3 \cap (K_1 \cup K_2)$, or $K_2 \supseteq K_3 \cap (K_1 \cup K_2)$, which means that $K_3 \cap (K_1 \cup K_2)$ equals either $K_1 \cap K_3$ or $K_2 \cap K_3$. Since

$$\kappa_1^{\downarrow K_2 \cap K_1} = \kappa_2^{\downarrow K_2 \cap K_1},$$

due to property 2 of Lemma 3 $\kappa_1 \triangleright \kappa_2 = \kappa_2 \triangleright \kappa_1$ and therefore, using Lemma 1, $(\kappa_1 \triangleright \kappa_2)^{\downarrow K_1} = \kappa_1$ and $(\kappa_1 \triangleright \kappa_2)^{\downarrow K_2} = \kappa_2$. Thus we got that $(\kappa_1 \triangleright \kappa_2)^{\downarrow K_3 \cap (K_1 \cup K_2)}$ equals either $\kappa_1^{\downarrow K_3 \cap K_1}$, or $\kappa_2^{\downarrow K_3 \cap K_2}$. Analogously, we can see that for any $r = 3, \ldots, m$ there exists s such that

$$(\kappa_1 \triangleright \ldots \triangleright \kappa_{r-1})^{\downarrow K_r \cap (K_1 \cup \ldots \cup K_{r-1})} = \kappa_s^{\downarrow K_r \cap K_s},$$

which makes the perfectization process computationally simple.

Another advantageous property concerning the RIP generating sequences is expressed in the following theorem.

Theorem 4. *If $\pi_1, \ldots, \pi_m$ is a sequence of pairwise consistent probability distributions such that $K_1, \ldots, K_m$ meets RIP, then this sequence is perfect.*

3 Conditioning

Define a *degenerated* one-dimensional probability distribution $\nu_{|i;a}$ as a distribution of variable X_i achieving probability 1 for value $X_i = a$, i.e.,

$$\nu_{|i;a}(X_i = x) = \begin{cases} 1 & \text{if } x = a,\\ 0 & \text{otherwise.} \end{cases}$$

Now, compute $(\nu_{|i;a} \triangleright \kappa)^{\downarrow \{X_j\}}$ for a probability distribution $\kappa(L)$ with $X_i, X_j \in L$. For $y \in \mathbb{X}_j$

$$\begin{aligned}
(\nu_{|i;a} \triangleright \kappa)^{\downarrow \{X_j\}}(y) &= ((\nu_{|i;a} \triangleright \kappa)^{\downarrow \{X_j, X_i\}})^{\downarrow \{X_j\}}(y)\\
&= (\nu_{|i;a} \triangleright \kappa^{\downarrow \{X_j, X_i\}})^{\downarrow \{X_j\}}(y)\\
&= \sum_{x \in \mathbb{X}_i} (\nu_{|i;a} \triangleright \kappa^{\downarrow \{X_j, X_i\}})(y, x)\\
&= \sum_{x \in \mathbb{X}_i} \frac{\nu_{|i;a}(x) \cdot \kappa^{\downarrow \{X_j, X_i\}}(y, x)}{\kappa^{\downarrow \{X_i\}}(x)}\\
&= \frac{\kappa^{\downarrow \{X_j, X_i\}}(y, a)}{\kappa^{\downarrow \{X_i\}}(a)}\\
&= \kappa^{\downarrow \{X_j, X_i\}}(y|a).
\end{aligned}$$

Repeating this computation we can easily deduce that for $\kappa(L)$

$$\kappa(L \setminus \{X_i\} | X_i = a) = (\nu_{|i;a} \triangleright \kappa)^{\downarrow L \setminus \{X_i\}},$$

and for $E = \{X_{i_1}, X_{i_2}, \ldots, X_{i_1}\}$, and $e \in \mathbb{X}_E$,

$$\begin{aligned}
&\kappa(L \setminus E | E = e)\\
&\quad = (\nu_{|i_1;e^{\downarrow \{X_{i_1}\}}} \triangleright (\ldots \triangleright (\nu_{|i_k;e^{\downarrow \{X_{i_k}\}}} \triangleright \kappa)))^{\downarrow L \setminus E}.
\end{aligned}$$

Now, we want to show that the last expression can be effectively computed for a multidimensional distribution κ represented in a form of a RIP perfect generating sequence: $\kappa = \pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m$.

Let us, first, consider the computation

$$\nu_{|i_k;e^{\downarrow \{X_{i_k}\}}} \triangleright \kappa = \nu_{|i_k;e^{\downarrow \{X_{i_k}\}}} \triangleright (\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m).$$

It is well known (and the reader can see it from the properties of join trees described in Section 4.1) that for each K_i the RIP sequence $K_1, K_2, \ldots, K_m$ can be reordered so that

the resulting sequence is also RIP and K_i is at the beginning of this new sequence. So, without loss of generality we can assume that $\pi_1, \pi_2, \ldots, \pi_m$ is such that $X_{i_k} \in K_1$. The fact that the respective reordering does not influence the represented multidimensional distribution $\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m$ follows from Theorems 4 and 2.

Now, consider

$$\nu_{|i_k;e^{\downarrow\{X_{i_k}\}}} \triangleright \big((\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_{n-1}) \triangleright \pi_m\big).$$

Since X_{i_k} is contained in K_1, the more it is contained in $(K_1 \cup K_2 \cup \ldots \cup K_{m-1})$, and therefore, applying property 4 of Lemma 3, we get

$$\begin{aligned}\nu_{|i_k;e^{\downarrow\{X_{i_k}\}}} &\triangleright \big((\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_{m-1}) \triangleright \pi_m\big) \\ &= \big(\nu_{|i_k;e^{\downarrow\{X_{i_k}\}}} \triangleright \big((\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_{m-1})\big)\big) \triangleright \pi_m.\end{aligned}$$

Repeating this reasoning $m-1$ times we eventually get

$$\begin{aligned}\nu_{|i_k;e^{\downarrow\{X_{i_k}\}}} &\triangleright (\pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m) \\ &= \big(\nu_{|i_k;e^{\downarrow\{X_{i_k}\}}} \triangleright \pi_1\big) \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m,\end{aligned}$$

which means that computing a conditional from a distribution represented by a RIP perfect generating sequence results, again, in a distribution represented as a RIP sequence. The latter can be, as showed in the preceding section, efficiently transformed into a RIP perfect generating sequence. Therefore, to compute $\kappa(L \setminus E|E = e)$ we have to successively apply the above idea for all $X_{i_j} \in E$. Remember that for each X_{i_j} we have to find a RIP ordering of $(K_1, K_2, \ldots, K_m)$ such that $X_{i_j} \in K_1$. Generally, for different X_{i_j} we have to use a different RIP ordering of $(K_1, K_2, \ldots, K_m)$.

4 Sequential Decision Process

A sequential decision process consists in a successive repetition of the following step:

> *Knowing values $e \in \mathbb{X}_E$ of variables E, find a variable $X_i \in N \setminus (E \cup \{X_\delta\})$ such that the detection of its value increases (as much as possible) the chances of getting*
>
> $$\kappa(X_\delta = d|E = e, X_i = a) \geq \varepsilon$$
>
> *for some value $d \in \mathbb{X}_\delta$.*

It is important to realize that in this step we search for a variable X_i whose value is to be established next. It means that at the moment of looking for X_i we do not know the value $a \in \mathbb{X}_i$. This value will be ascertain before the next sequential step is realized with new $E := E \cup \{X_i\}$.

For the selection of $X_i \in N \setminus (E \cup \{X_\delta\})$ we can hardly find a better criterion than that used in the process of construction of efficient decision (or search) trees for many years [Knuth, 1971; Jiroušek, 1975; Quinlan, 1990]. Using this criterion, in the process of data based construction of a decision tree, we look for the variable that splits the considered training data file into subfiles yielding the minimum expected Shannon entropy for the decision variable. In fact, it is nothing else than looking for a variable $X_i \in N \setminus (E \cup \{X_\delta\})$ maximizing the Shannon conditional mutual information

$$\begin{aligned}&MI_\kappa(X_i; X_\delta|E = e) \\ &= \sum_{(x,y)\in\mathbb{X}_i\times\mathbb{X}_\delta} \kappa^{\downarrow\{X_i,X_\delta\}}(x, y|E = e) \\ &\qquad \cdot \log \frac{\kappa^{\downarrow\{X_i,X_\delta\}}(x, y|E = e)}{\kappa^{\downarrow\{X_i\}}(x|E = e)\kappa^{\downarrow\{X_\delta\}}(y|E = e)}.\end{aligned}$$

Now, as the reader certainly expects, we take advantage of the results from the preceding section and assume that the conditional distribution $\kappa(N \setminus E|e)$ is represented in a form of a RIP perfect generating sequence. This assumption enables us not only to compute values

$$MI_\kappa(X_j; X_\delta|E = e) = MI_{\kappa(N\setminus E|e)}(X_j; X_\delta)$$

for all $X_j \in N \setminus (E \cup \{X_\delta\})$ in an efficient way, but also to speed up the computational process by indicating those variables, for which we need not compute the value of mutual information because we can learn in advance that the respective conditional mutual information cannot achieve a maximal value.

4.1 Computation in Join Trees

It is known from both data-base [Beeri *et al.*, 1983] and Bayesian network [Jensen and Nielsen, 2007] theories that a system of sets $K_1, K_2, \ldots, K_m$ can be ordered to meet RIP if and only if one can construct a structure called a *join tree*. It is a tree having $K_1, K_2, \ldots, K_m$ for its nodes and possessing the following special property: If K_k lies on the path from K_r to K_s then $K_k \supseteq K_r \cap K_s$.

Recall that it is an easy task to construct a join tree for a RIP sequence $K_1, K_2, \ldots, K_m$: For, each K_r ($r = 2, \ldots, m$) the join tree contains an edge connecting K_r with that K_s, which meets the RIP condition

$$1 \leq s < r \ \ \& \ \ K_r \cap (K_1 \cup \ldots \cup K_{r-1}) \subseteq K_s.$$

If there are more such nodes K_s, then K_r is connected to only one of them. The tree contains no other edges than those specified above.

To compute $MI_\kappa(X_j; X_\delta|E = e)$ for all $X_j \in N \setminus (E \cup \{X_\delta\})$, start with enumerating this mutual information for all the variables from K_r, for which $X_\delta \in K_r$ (if there are more sets meeting this condition, consider all of them). Since we assume that

$$\kappa(N \setminus E|e) = \pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m,$$

and that $\pi_1, \pi_2, \ldots, \pi_m$ is a perfect sequence, due to Theorem 1 we know that π_r is marginal of $\kappa(N \setminus E|e)$. This means that we can compute the required conditional mutual information just from π_r, which is simple.

After having evaluated the required mutual information for all the variables from $K_r \setminus \{X_\delta\}$, we start processing variables from the neighboring nodes, i.e., from nodes K_s that are adjacent to K_r in the join tree. Now, it is important to realize two facts that makes the computation very efficient.

First, since K_s is adjacent to K_r in the join tree, it is possible to find a RIP ordering of $K_1, K_2, \ldots, K_m$ such that it starts $K_r, K_s, ...$, and therefore (thanks to Theorem 2 and Lemma 1) we know that

$$\kappa((K_r \cup K_s) \setminus E|e) = \pi_r \triangleright \pi_s.$$

The other fact that can speed up the computational process follows from a famous property of mutual information. It is known from any textbook on information theory (e.g. [Hamming, 1986]) that if $X_j \perp\!\!\!\perp X_\delta | M[\mu]$, for some set of variables M then

$$MI_\mu(X_j; X_\delta) \leq MI_\mu(M; X_\delta).$$

Therefore, applying this property and Lemma 2 to this situation we get that for $X_j \in K_s \setminus K_r$ (recall that $X_\delta \in K_r \setminus K_s$ because all variables from K_r were treated in the previous step)

$$\begin{aligned} MI_\kappa(X_j; X_\delta | E = e) &= MI_{\pi_r \triangleright \pi_s}(X_j; X_\delta) \\ &< MI_{\pi_r \triangleright \pi_s}(M; X_\delta). \end{aligned}$$

This means that if there is variable $X_i \in K_r$ for which

$$MI_{\pi_r \triangleright \pi_s}(K_r \cap K_s; X_\delta) \leq MI_\kappa(X_i; X_\delta | E = e)$$

then we do not need to compute mutual information $MI_{\pi_r \triangleright \pi_s}(X_j; X_\delta)$ for variables $X_j \in K_s \setminus K_r$ because we know that it cannot achieve the looked for maximum.

In a similar way we can compute $MI_\kappa(X_j; X_\delta | E = e)$ for all the remaining variables from $N \setminus (E \cup \{X_\delta\})$. First we have to find the shortest path in the considered join tree connecting nodes containing X_j and X_δ (realize that in a tree, two nodes are always connected by a unique path, but both the considered variables X_j and X_δ may be in several nodes). Denote this path $K_{j_1}, K_{j_2}, \ldots, K_{j_k}$ in the way that $X_j \in K_{j_1}$ and $X_\delta \in K_{j_k}$. Then from the perfectness of $\pi_1, \pi_2, \ldots, \pi_m$ and the RIP property we get that

$$\kappa((K_{j_1} \cup K_{j_2} \cup \ldots \cup K_{j_k}) \setminus E | e) = \pi_{j_1} \triangleright \pi_{j_2} \triangleright \ldots \triangleright \pi_{j_k},$$

and thus we compute

$$MI_\kappa(X_j; X_\delta | E = e) = MI_{\pi_{j_1} \triangleright \pi_{j_2} \triangleright \ldots \triangleright \pi_{j_k}}(X_j; X_\delta).$$

However the longer path $K_{j_1}, K_{j_2}, \ldots, K_{j_k}$ the greater chances that among the variables, for which the mutual information has been evaluated we can find variable X_i, for which

$$MI_{\pi_{j_1} \triangleright \pi_{j_2} \triangleright \ldots \triangleright \pi_{j_k}}(K_1 \cap K_2; X_\delta) \leq MI_\kappa(X_i; X_\delta | E = e),$$

and therefore the computation of
$MI_{\pi_{j_1} \triangleright \pi_{j_2} \triangleright \ldots \triangleright \pi_{j_k}}(X_j; X_\delta)$ is wasteful.

4.2 An Algorithm

Up to now, we have described and theoretically supported all the individual parts of the proposed sequential decision process. Let us (rather informally) summarize it in several steps.

Initialization

We assume that the probability distribution represented by a RIP generating sequence is given. If it is not perfect then apply the perfectization procedure described in Theorem 3 so that the resulting generating sequence $\pi_1, \pi_2, \ldots, \pi_m$ meets RIP condition and is perfect. Thus, in the sequel we compute with the distribution

$$\kappa(N) = \pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m.$$

Set $E := \emptyset$.

Preliminary evidence processing

E_0 denotes the set of variables whose values e_0 are given before the sequential process starts. If there is no preliminary evidence, i.e. $E_0 = \emptyset$, skip the rest of this step. Otherwise for all variables X_i from E_0 realize the following conditioning procedure.

Conditioning. Assign a new value to $E := E \cup \{X_i\}$, and extend the point $e := e_0^{\downarrow E}$. Reorder (renumber) the given generating sequence in the way that the new ordering $K_1, K_2, \ldots, K_m$ meets RIP and $X_i \in K_1$. Apply the perfectization procedure (Theorem 3) to the sequence: $(\nu_{|j; e_0^{\downarrow X_i}} \triangleright \pi_1), \pi_2, \ldots, \pi_m$, and the result, after marginalizing variable X_i out, assign as a new value to $\pi_1, \pi_2, \ldots, \pi_m$. So that now,

$$\kappa(N \setminus E | e) = \pi_1 \triangleright \pi_2 \triangleright \ldots \triangleright \pi_m.$$

Sequential procedure

Perform the following process consisting of three steps (**Variable selection**, **Application** and **Conditioning**) repeatedly until $\pi_r^{\downarrow \{X_\delta\}}(d) \geq \varepsilon$ for some $d \in \mathbb{X}_\delta$. (Take any r such that $X_\delta \in K_r$.)

Variable selection. Set $L := N \setminus (E \cup \{X_\delta\})$ (L is a set of variables, for which we should compute the value of mutual information in this step).
For all $X_j \in L$, for which there exists $r \in \{1, 2, \ldots, m\}$ such that both $X_j, X_\delta \in K_r$, compute

$$MI_\kappa(X_j; X_\delta | E = e) = MI_{\pi_r}(X_j; X_\delta),$$

and reset $L := L \setminus \{X_j\}$.
Denote X_i the variable achieving the maximal value of the mutual information.
Repeat the following step until $L = \emptyset$.

Computation in a join tree. Find K_{j_1}, such that the path $K_{j_1}, K_{j_2}, \ldots, K_{j_k}$ from the respective join tree meets the following three properties: (1) $X_\delta \in K_{j_k}$, (2) $K_{j_1} \cap L \neq \emptyset$, and (3) $(K_{j_2} \cup \ldots \cup K_{j_k}) \cap L = \emptyset$.

If

$$MI_{\pi_{j_2} \triangleright \ldots \triangleright \pi_{j_k}}(K_{j_1} \cap K_{j_2}; X_\delta) > MI_\kappa(X_i; X_\delta | E = e)$$

Then
For all $j \in K_{j_1} \cap L$ compute

$$MI_\kappa(X_j; X_\delta | E = e) = MI_{\pi_{j_1} \triangleright \ldots \triangleright \pi_{j_k}}(X_j; X_\delta),$$

and reset $L := L \setminus K_{j_1}$.
If the maximum from the computed values of mutual information is higher than

$$MI_\kappa(X_i; X_\delta | E = e)$$

then reset X_i to the variable with the highest mutual information.

Else
Reset L by removing from L all such K_s for which the path from K_s to K_{j_k} goes through K_{j_1}.

Application. Ask the user to ascertain the value of variable X_i. Denote the result a.

Conditioning. Reset $E := E \cup \{X_i\}$, and extend the point e so that the new value is from $\mathbb{X}_E$, and $e^{\downarrow\{X_i\}} = a$. Reorder (renumber) generating sequence $\pi_1, \pi_2, \ldots, \pi_m$ in the way that the respective new ordering $K_1, K_2, \ldots, K_m$ meets RIP and $X_i \in K_1$. Apply the perfectization procedure (Theorem 3) to the sequence: $(\nu_{|i;e^{\downarrow x_i}} \rhd \pi_1), \pi_2, \ldots, \pi_m$, and the result, after marginalizing variable X_i out, assign as a new value to $\pi_1, \pi_2, \ldots, \pi_m$. So that now,

$$\kappa(N \setminus E|e) = \pi_1 \rhd \pi_2 \rhd \ldots \rhd \pi_m.$$

5 Summary and Conclusions

We have described a sequential diagnosis making process based on a knowledge represented by a multidimensional probability distribution. The reader certainly realized that the controlling rule aims for the least number of variables whose values are to be ascertain. As said in Introduction, it is really not difficult to modify the variable selection rule so that some weights of the variables are taken into account. In this case, however, one can hardly rely upon the fact that the longer path $K_{j_1}, K_{j_2}, \ldots, K_{j_k}$ is constructed in the **Computation in a join tree** step the greater chances to cut off the nodes of the join tree that cannot contain a variable optimizing the selection criterion.

As it can be seen from the previous text, the application of perfect compositional models have one great advantage visible in comparison with Bayesian networks. Each low-dimensional distribution, which the model is constructed from, is marginal to the multidimensional model. This makes verification of some conditions, like for example the stoping rule from the Algorithm, very simple. More generally, there is a whole class of sets of variables, for which the marginals can easily be computed. Theses are the sets that can be got as a union of nodes of a subtree (connected subgraph) of the respective join tree. This property was exploited in the **Computation in a join tree** step of the Algorithm, where we considered marginals $\pi_{j_1} \rhd \ldots \rhd \pi_{j_k}$.

Though it is beyond the scope of this paper, let us mention yet another advantage of the considered compositional models. The operator of composition was introduced also in possibility theory [Vejnarová, 1998] and recently even in Shenoy's Valuation Based Systems [Shenoy, 1992; Jiroušek and Shenoy, 2013], which, as a generic uncertainty calculus covers many other calculi, such as Spohn's epistemic belief theory, and D-S belief function theory. This makes it possible to apply compositional models, and all the methods based on the compositional models like the described sequential decision process, in all these alternative uncertainty theories.

References

[Beeri *et al.*, 1983] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[Hamming, 1986] Richard W Hamming. *Coding and information theory*. Prentice-Hall, Inc., 1986.

[Jensen and Nielsen, 2007] Finn Verner Jensen and Thomas Dyhre Nielsen. *Bayesian networks and decision graphs*. Springer, 2007.

[Jiroušek and Shenoy, 2013] Radim Jiroušek and Prakash P Shenoy. Compositional models in valuation-based systems. *International Journal of Approximate Reasoning*, 2013. doi: 10.1016/j.ijar.2013.02.002.

[Jiroušek, 1997] Radim Jiroušek. Composition of probability measures on finite spaces. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 274–281. Morgan Kaufmann Publishers Inc., 1997.

[Jiroušek, 2000] Radim Jiroušek. Marginalization in composed probabilistic models. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 301–308. Morgan Kaufmann Publishers Inc., 2000.

[Jiroušek, 1975] Radim Jiroušek. Heuristic methods of construction of sequential questionnaire. *Kybernetika*, 11(4):253–270, 1975.

[Jiroušek, 2011] Radim Jiroušek. Foundations of compositional model theory. *International Journal of General Systems*, 40(6):623–678, 2011.

[Knuth, 1971] Donald E Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971.

[Lauritzen, 1996] Steffen L Lauritzen. *Graphical models*. Oxford University Press, 1996.

[Quinlan, 1990] John Ross Quinlan. Decision trees and decision-making. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(2):339–346, 1990.

[Shenoy, 1992] Prakash P Shenoy. Valuation-based systems for bayesian decision analysis. *Operations research*, 40(3):463–484, 1992.

[Studený, 2005] Milan Studený. *Probabilistic conditional independence structures*. Springer, 2005.

[Vejnarová, 1998] J Vejnarová. Possibilistic independence and operators of composition of possibility measures. In *Prague conference on information theory, statistical decision functions and random processes*, pages 575–580. JČMF, 1998.

Distributed Analysis for Diagnosability in Concurrent Systems *

Hernán Ponce de León[1] and **Gonzalo Bonigo**[2] and **Laura Brandán Briones**[2,3]
[1]INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France
[2]Fa.M.A.F. - Universidad Nacional de Córdoba, Argentina
[3]CONICET
e-mail: ponce@lsv.ens-cachan.fr, bonigo@famaf.unc.edu.ar, lbrandan@famaf.unc.edu.ar

Abstract

Complex systems often exhibit unexpected faults that are difficult to handle. Such systems are desirable to be diagnosable, i.e. faults can be automatically detected as they occur (or shortly afterwards), enabling the system to handle the fault or recover. A system is diagnosable if it is possible to detect every fault, in a finite time after they occurred, by only observing the available information from the system. Complex systems are usually built from simpler components running concurrently. We study how to infer the diagnosability property of a complex system (distributed and with multiple faults) from a parallelized analysis of the diagnosability of each of its components synchronizing with fault free versions of the others. In this paper we make the following contributions: (1) we address the diagnosability problem of concurrent systems with arbitrary faults occurring freely in each component. (2) We distribute the diagnosability analysis and illustrate our approach with examples. Moreover, (3) we present a prototype tool that implements our techniques showing promising results.

1 Introduction

As systems become larger, their behavior becomes more complex. Several things may go wrong, resulting in faults occurring. It is then crucially important to design our systems in a way that we can detect or recover from such faults when they occur. A system is diagnosable when its design allows the detection of faults, for instance a system that has sensors specially dedicated to detect them. Sometimes the detection of faults is more involved and the diagnosability property is harder to establish, specially in systems with several components.

A sound software engineering rule for building complex systems is to divide the whole system in smaller and simpler components, each solving a specific task. Moreover, usually they are built by different groups of people and may be in different places. This means that, in general, complex systems are actually collections of simpler components running in parallel.

*This work has been supported by the European Union Sh Framework Programme under grant agreement no. 295261 (MEALS).

In order to model such systems and formally prove results, there are several formalisms like Finite State Machines (FSMs) [Sampath *et al.*, 1995; Jiang *et al.*, 2000], Petri Nets [Genc and Lafortune, 2003; Madalinski *et al.*, 2010] and Labeled Transition Systems (LTSs) [Brandán-Briones *et al.*, 2008; Brandán-Briones and Madalinski, 2011; Bonigo and Brandán-Briones, 2012]. In this paper, we model each component by a LTS, so the whole system is a collection of LTSs synchronizing in all their shared observable actions (see Section 2).

In the diagnosability analysis of distributed systems it is usually assumed that a fault can occur in exactly one of the different components. We relax this assumption allowing the same fault to occur in several components.

Also, the diagnosability analysis is usually iterative (i.e., sequential): the information from local diagnosers is combined until a global verdict is reached. We propose a method to distribute this analysis.

Finally, we developed a tool that implements all our research. The DADDY tool (Distributed Analysis for Distributed Discrete sYstems) [Bonigo, 2012] is a prototype based on the results presented in [Bonigo and Brandán-Briones, 2012] and this paper. The tool does not only implement the method we presents but also the classic one allowing us to compare both approaches. We present a comparative analysis of their performance obtained from the experimental running of several examples.

Related Work Diagnosability was initially developed in [Sampath *et al.*, 1995] under the setting of discrete event systems. In that paper, necessary and sufficient conditions for testing diagnosability are given. In order to test diagnosability, a special diagnoser is computed, whose complexity of construction is shown to be exponential in the number of states of the original system, and double exponential in the number of faults. Later, in [Jiang *et al.*, 2000], an improvement of this algorithm is presented, where the so-called twin plant method is introduced and shown to have polynomial complexity in the number of states and faults. Afterwards, in [Schumann and Pencolé, 2007], an improvement to the twin plant method is presented where the system is reduced before building the twin plant.

None of the methods presented there (i.e., [Sampath *et al.*, 1995; Jiang *et al.*, 2000]) consider the problem when the system is composed of components working in parallel. An approach to this consideration is addressed in [Schumann and Pencolé, 2007; Debouk *et al.*, 2000; Pencolé, 2004; Schumann and Huang, 2008] where the diagnosability prob-

lem is performed by either local diagnosers or twin plants communicating with each other, directly or through a coordinator, and by that means pooling together the observations. [Ye and Dague, 2012] shows that when considering only local observations, diagnosability becomes undecidable when the communication between component is unobservable. An algorithm is proposed to check a sufficient but not necessary condition of diagnosability. However, their results are based in the assumption that a fault can only occur in one of the components, an assumption that can not always be made.

Several mechanisms such as interleaving, shared variables and handshaking have been described in [Baier and Katoen, 2008] to provide operational models for distributed systems. In the handshaking method, the communication is made by the synchronization on actions or events. These actions must be specified a priori in the model, so the different components can be synchronized at execution time. In [Bonigo and Brandán-Briones, 2012] the authors study how different kinds of synchronizations (via all the shared actions, some of them or none) impact in the diagnosis analysis.

Motivation Suppose different groups of people are commanded to build different components of a system. Even if each component is diagnosable, it is not always the case that the resulting system has such property[1]. In [Bonigo and Brandán-Briones, 2012] the authors show that with different kinds of synchronizations, the diagnosability of the global system can not be inferred directly from the diagnosability of each component.

We propose a framework where each component only shares with the rest a fault free version of its own, maybe the specification of its ideal behavior. Then, each component should not only be diagnosable, but also its interaction with the fault free version of the others, i.e. its synchronous product with fault free version of the other components. Therefore, our diagnosability analysis can be distributed.

Paper organization Section 2 presents the formal model that we use for modeling each component, the parallel composition between them and the notion of diagnosability. In Section 3, we develop our analysis method, showing how the diagnosability of each component synchronizing with fault free versions of the other components influences the diagnosability property of the overall system. Section 4 presents our tool DADDY and some experimental results. We conclude and discuss about future work in Section 5.

2 Diagnosability Analysis

2.1 Model of the system

We consider a distributed system composed of two autonomous components G_1, G_2 that communicate with each other by all their shared observable actions. The local model of a component is defined as a Labeled Transition System.

Definition 1. *A Labeled Transition System (LTS) is a tuple* $G = (Q, \Sigma, \delta, q_0)$ *where*

- Q *is a finite set of states,*
- Σ *is a finite set of actions,*
- δ *is a partial transition function, and*
- q_0 *the initial state, with* $q_0 \in Q$.

As usual in diagnosability analysis, some of the actions of Σ are observable while the rest are unobservable. Thus the set of actions Σ is partitioned as $\Sigma = \Sigma_o \uplus \Sigma_{uo}$ where Σ_o represents the observable actions and Σ_{uo} the unobservable ones.

The faults to diagnose are considered unobservable, i.e. $\Sigma_F \subseteq \Sigma_{uo}$, as faults that are observable can be easily diagnosable.

As usual in diagnosability analysis, we made the following assumptions about our systems.

Assumption 1. *We only consider (live) systems where there is a transition defined at each state, i.e. the system cannot reach a point at which no action is possible.*

Assumption 2. *The system does not contain cycles of unobservable actions.*

Note that, these assumptions together assure that all our systems are free of observation starvation.

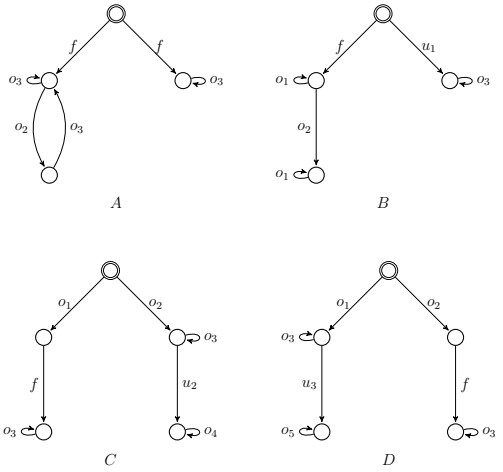


Figure 1: Specification of four components modeled by LTSs

Figure 1 shows four components modeled by the LTSs A, B, C and D where $o_1, o_2, o_3, o_4, o_5 \in \Sigma_o$ and $u_1, u_2, u_3 \in \Sigma_{uo}$. The special action $f \in \Sigma_F$ is the fault to be diagnosable.

A path from state q_i to state q_j in G is a sequence $q_i \cdot a_i \cdot q_{i+1} \ldots a_{j-1} \cdot q_j$ such that $(q_k, a_k, q_{k+1}) \in \delta$ for $i \leq k \leq j-1$. The set of paths in G is denoted by paths(G).

The trace associated with any given path consists of its sequence of actions (i.e., for a path $\rho = q_0 \cdot a_0 \cdot q_1 \ldots a_{n-1} \cdot q_n$ we have trace$(\rho) = a_0 \cdot a_1 \ldots a_n$). Given a trace, $\sigma = a_0 \cdot a_1 \ldots a_n$, we denote as $f \in \sigma$ when there exists i such that $f = a_i$. As our systems are live, we only consider infinity traces where the infinite repetition of an actions a is denoted by $\widehat{a}$. The set of all traces starting in q_0 is denoted by traces(G). As we consider nondeterministic systems, the same trace can belong to several paths. The set of possible paths of a trace σ in G are: path$(\sigma) = \{\rho \in \text{paths}(G) \mid \text{trace}(\rho) = \sigma\}$.

The observation of a trace is given by the following definition.

[1] See for example C, D and $C \times D$ in Figures 1 and 2.

Definition 2. *Let $\sigma \in \Sigma^*$, then*

$$obs(\sigma) \quad = \quad \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a \cdot obs(\sigma') & \text{if } \sigma = a \cdot \sigma' \wedge a \in \Sigma_o \\ obs(\sigma') & \text{if } \sigma = a \cdot \sigma' \wedge a \notin \Sigma_o \end{cases}$$

The communication between two components is given by their synchronous product where the synchronizing actions are all the shared observable ones.

Definition 3. *Given two local components $G_1 = (Q^1, \Sigma^1, \delta^1, q_0^1)$ and $G_2 = (Q^2, \Sigma^2, \delta^2, q_0^2)$, the behavior of the global system is given by their synchronous product $G_1 \times G_2 = (Q^1 \times Q^2, \Sigma^1 \cup \Sigma^2, \delta^{1\times 2}, (q_0^1, q_0^2))$ where $\delta^{1\times 2}$ is defined as follows*

$$\delta^{1\times 2}((q_i^1, q_j^2), a) = \begin{cases} (\delta^1(q_i^1, a), \delta^2(q_j^2, a)) & \text{if } a \in \Sigma_o^1 \cap \Sigma_o^2 \\ (\delta^1(q_i^1, a), q_j^2) & \text{if } a \in \Sigma^1 \wedge a \notin \Sigma^2 \\ (q_i^1, \delta^2(q_j^2, a)) & \text{if } a \in \Sigma^2 \wedge a \notin \Sigma^1 \end{cases}$$

Given a path in the global system, we can project it to a single component.

Definition 4. *Let $\rho \in paths(G_1 \times G_2)$, its projection in G_i is*

$$P_i((q^1, q^2)) \quad = \quad q^i$$

$$P_i((q^1, q^2) \cdot a \cdot \rho') \quad = \quad \begin{cases} q^i \cdot a \cdot P_i(\rho') & \text{if } \exists\, \delta^i(q^i, a) \\ P_i(\rho') & \text{otherwise} \end{cases}$$

For a trace in the global system, we define the projections to know which actions belong to a certain component.

Definition 5. *Let σ be a trace in traces$(G_1 \times G_2)$, σ' is its projection in G_i, denoted $P_i(\sigma) = \sigma'$, iff*

$$\exists\, \rho \in path(\sigma) : trace(P_i(\rho)) = \sigma'$$

Example 1. *Let $\sigma = o_1 f o_3 u_3 \widehat{o_5}$ be a trace in traces$(C \times D)$ from Figure 2, its projection in component C is given by $P_C(\sigma) = o_1 f o_3$ and its projection in component D is given by $P_D(\sigma) = o_1 o_3 u_3 \widehat{o_5}$. These projections are traces of the corresponding components C and D from Figure 1. Note that projections of an infinite trace from the global system can be finite in one of the components.*

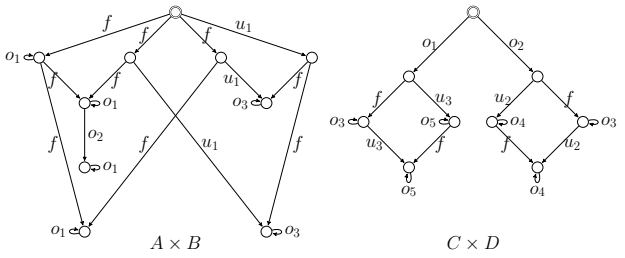


Figure 2: Synchronous product of components A, B and C, D

As the projection operator only erases actions in a trace, it is easy to see that every fault belonging to a projection of such a trace, also belongs to the trace in the global system as it is shown by the following result.

Proposition 1. *For every trace σ in traces$(G_1 \times G_2)$ with $P_i(\sigma) = \sigma_i$, we have*

$$\text{if} \quad f \in \sigma_i \quad \text{then} \quad f \in \sigma$$

When two components synchronize in all their shared actions, if two traces of the global system have the same observability and we project them to the same component, the resulting projections will also have the same observability. This result is captured by Proposition 2.

Proposition 2. *Given two traces σ and α in traces$(G_1 \times G_2)$ with $P_i(\sigma) = \sigma_i$ and $P_i(\alpha) = \alpha_i$, we have*

$$\text{if } obs(\sigma) = obs(\alpha) \quad \text{then} \quad obs(\sigma_i) = obs(\alpha_i)$$

This result is proved by double induction in the structure of σ and α. We analyze several cases depending on the existence of the projections. One of the most critical cases is when $\sigma = a \cdot \sigma', \alpha = b \cdot \alpha', a \in \Sigma_o^1 \cap \Sigma_o^2$, but $b \notin \Sigma_o^1 \cap \Sigma_o^2$ as it has several particular sub-cases. Note that this result only holds when the synchronization is done in all the set of shared actions.

2.2 Diagnosability condition

We present now the notion of diagnosability. Informally, a fault $f \in \Sigma_F$ is diagnosable if it is possible to detect, within a finite delay, occurrences of such a fault using the record of observed actions. In other words, a fault is not diagnosable if there exist two infinite paths from the initial state with the same infinite sequence of observable actions but only one of them contains a fault.

Definition 6. *Let f be a fault in Σ_F, f is diagnosable in G iff*

$$\forall \sigma, \alpha \in traces(G) : \text{if } obs(\sigma) = obs(\alpha)$$
$$\text{and } f \in \sigma \text{ then } f \in \alpha$$

*The system G is diagnosable, denoted by **diag(G)**, if and only if every fault $f \in \Sigma_F$ is diagnosable.*

The previous definition introduced in [Brandán-Briones *et al.*, 2008] is a reformulation of the one presented in [Sampath *et al.*, 1995].

Example 2. *Let consider the components A and B from Figure 1. The only pair of traces in A with the same observability are of the form $f\widehat{o_3}$ (one for each branch from the initial state), as both traces contain the fault f, system A is diagnosable. In the case of B, each observable trace corresponds to a unique path, therefore B is diagnosable.*

Now, consider system $A \times B$ from Figure 2, we can see that every trace contains a fault, therefore $A \times B$ is diagnosable. On the contrary, in system $C \times D$ we have two traces, $o_2 u_2 \widehat{o_4}$ and $o_2 f u_2 \widehat{o_4}$ that have the same observability, but one of them contains a fault and the other does not, therefore the system $C \times D$ is not diagnosable.

3 Distributing the diagnosability analysis

The notion of diagnosability is introduced in [Sampath *et al.*, 1995] assuming a centralized architecture of the system. In order to check the diagnosability property in distributed systems, the synchronous product of components is computed and such a product is given as an input to an algorithm that tests its diagnosability (usually based on the twin plant method). The size of such a product grows exponentially with respect of the size of the components, resulting in an inefficient algorithm. When dealing with real applications, such as telecommunication networks or power distribution networks, the centralized approach is clearly unrealistic because of the size of those applications. Moreover, this approach does not exploit the fact that such systems are distributed.

In [Schumann and Pencolé, 2007; Pencolé, 2004] the authors distribute the search for non-distinguishable behaviors based on local verifiers and local twin plants. The local information is propagated until a verdict is made or, in the

worst case, the global system is built. Their result is based on the assumption that a fault can occur in exactly one component.

In this section we present a method that allows to decide the diagnosability of a distributed system in terms of the diagnosability of each faulty component synchronizing with fault free versions of the remaining ones. Basically, we compose each component with a fault free version of the other components and analyze their diagnosability in parallel. To the best of our knowledge, it is the first method that allows to parallelize the diagnosability analysis.

Algorithm 1

Require: A LTS $G = (Q, \Sigma, \delta, q_0)$
Ensure: An f-fault free version of G
1: $Q' := \{q_0\}$, $\delta' := \emptyset$, $Q := Q \setminus \{q_0\}$
2: **while** $\exists (q',x,q) : q' \in Q' \wedge (q',x,q) \in \delta \wedge (q',x,q) \notin \delta'$ **do**
3: **if** $x \neq f$ **then**
4: $Q' := Q' \cup \{q\}$
5: $\delta' := \delta' \cup (q', x, q)$
6: **end if**
7: $\delta := \delta \setminus (q', x, q)$
8: **end while**
9: **return** $G^f = (Q', \Sigma, \delta', q_0)$

For testing the diagnosability of a fault $f \in \Sigma_F$ in the global system, instead of computing the whole composition, we consider one component and compose it with the fault free versions of the others. These fault free versions may be taken as the specification of each component, when provided, or can be computed by removing the fault f in the component using Algorithm 1 and considering such as the correct behavior of the system.

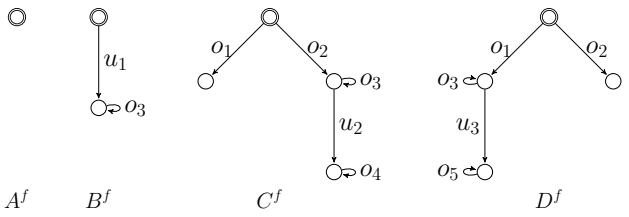


Figure 3: Components A, B, C and D after removing their faults

If we compose a component G_i with the fault free version of G_j, meaning G_j^f, clearly the traces of the resulting system are those of $G_i \times G_j$ such that its projections in G_j are fault free.

Proposition 3. *Let G_i and G_j be two LTSs, then $\sigma \in$ traces$(G_i \times G_j^f)$ iff*

$$\sigma \in \textit{traces}(G_i \times G_j) \wedge \forall \sigma_j : P_j(\sigma) = \sigma_j \Rightarrow f \notin \sigma_j$$

Figure 3 shows components A, B, C and D after removing fault f and Figure 4 shows them synchronizing with the faulty components.

Example 3. *Let us consider the systems from Figure 4. System $A^f \times B$ is trivially diagnosable. In the case of system $A \times B^f$, it is easy to see that the observable traces are of the form $\widehat{o_3}$, but all traces containing o_3 also contain f and therefore $A \times B^f$ is also diagnosable. In system $C^f \times D$, traces $\sigma = o_2 u_2 \widehat{o_4}$ and $\alpha = o_2 f u_2 \widehat{o_4}$ have the same observability, but α contains a fault and σ does not. So, we can conclude that $C^f \times D$ is not diagnosable.*

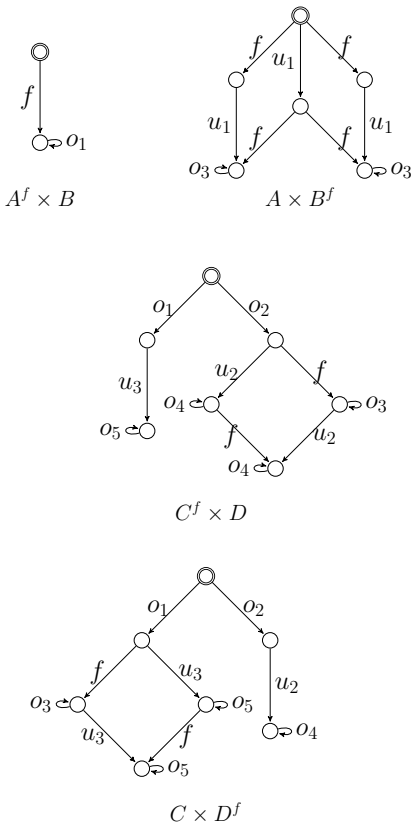


Figure 4: Composed systems after removing the faults in one of the components

The following result states necessary conditions for the diagnosability of the global system, i.e. the non diagnosability of $G_1^f \times G_2$ or $G_1 \times G_2^f$ implies the non diagnosability of $G_1 \times G_2$.

Theorem 1. *Let G_1 and G_2 be two LTSs, then*

$$\textbf{\textit{diag}}(G_1 \times G_2) \;\Rightarrow\; \textbf{\textit{diag}}(G_1^f \times G_2) \wedge \textbf{\textit{diag}}(G_1 \times G_2^f)$$

Proof. Lets assume that $\neg$**diag**$(G_1^f \times G_2)$, then there exist two traces $\sigma, \alpha \in$ traces$(G_1^f \times G_2)$ and f such that $obs(\sigma) = obs(\alpha)$ with $f \in \sigma$, but $f \notin \alpha$. We know from Proposition 3 that every trace in $G_1^f \times G_2$ is a trace in $G_1 \times G_2$, so we have found two traces of the global system with the same observability, one containing a fault and the other one not. Therefore $(G_1 \times G_2)$ is non-diagnosable. An analogous analysis can be made if $\neg$**diag**$(G_1 \times G_2^f)$. □

Example 4. *We see in Example 3 that $C^f \times D$ is non diagnosable. Using Theorem 1 we can conclude that $C \times D$ is non diagnosable. This result is consistent with the diagnosability analysis made in Example 2.*

As explained above, the idea is to build a diagnosable component and to test that its interaction with another fault free component is also diagnosable. We can then decide the

diagnosability of $G_1 \times G_2$ in term of the diagnosability of $G_1, G_2, G_1^f \times G_2$ and $G_1 \times G_2^f$.

Theorem 2. *Let G_1 and G_2 be two LTSs, then*

$$\left.\begin{array}{l} \textbf{\textit{diag}}(G_1) \wedge \textbf{\textit{diag}}(G_1 \times G_2^f) \\ \textbf{\textit{diag}}(G_2) \wedge \textbf{\textit{diag}}(G_1^f \times G_2) \end{array}\right\} \Rightarrow \textbf{\textit{diag}}(G_1 \times G_2)$$

Proof. Let assume that we have a fault $f \in \Sigma_F$ and two traces $\sigma, \alpha \in \text{traces}(G_1 \times G_2)$ with $f \in \sigma$ and $obs(\sigma) = obs(\alpha)$, we need to prove that $f \in \alpha$. Consider the following cases:

1. if $\sigma, \alpha \in \text{traces}(G_i^f \times G_j)$ we can prove by $(G_i^f \times G_j)$'s diagnosability that $f \in \alpha$ and then $G_1 \times G_2$ is diagnosable,
2. if $\alpha \notin \text{traces}(G_i^f \times G_j)$, using the hypothesis that $\alpha \in \text{traces}(G_i \times G_j)$, we can apply Proposition 3 and obtain that $\exists \alpha_i : P_i(\alpha) = \alpha_i \wedge f \in \alpha_i$. By Proposition 1 we know that every fault belonging to a projection also belongs to the trace in the global system, then $f \in \alpha$ and $G_1 \times G_2$ is diagnosable,
3. if $\alpha \in \text{traces}(G_i^f \times G_j)$ and $\sigma \notin \text{traces}(G_i^f \times G_j)$ we know by Proposition 3 that $\forall \alpha_i : P_i(\alpha) = \alpha_i$ and $f \notin \alpha_i$ and also that $\exists \sigma_i : P_i(\sigma) = \sigma_i$ with $f \in \sigma_i$. As $obs(\sigma) = obs(\alpha)$ we have that $obs(\sigma_i) = obs(\alpha_i)$ by Proposition 2. Finally as G_i is diagnosable and $f \in \sigma_i$, the fault should belong to α_i, leading to a contradiction. We can conclude that $G_1 \times G_2$ is diagnosable.

□

Example 5. *From Example 2 and Example 3 we know that $A, B, A^f \times B$ and $A \times B^f$ are diagnosable. If we apply Theorem 2 we can conclude that $A \times B$ is diagnosable, which is consistent with the analysis made in Example 2.*

3.1 Generalization

Until now we only consider systems composed by only two components. However, real examples are usually more complex and are composed of several components. Therefore we need to generalize the previous results to global systems composed of n different components running in parallel.

In order to generalize all our results, the associativity and commutativity property of synchronous product become essential. Note that in a general case the set of synchronizing actions is not necessarily the intersection of all their observable actions. Suppose that a system is composed by three components, G_1, G_2 and G_3, where two of them synchronize via an action a that does not belong to a third component, i.e. $a \in \Sigma_o^1 \cap \Sigma_o^2$, but $a \notin \Sigma_o^3$. We expect that G_1 and G_2 still synchronize in a. Fortunately, despite its apparent complications, the synchronous product is associative and commutative. The proof of such result can be found in previous work [Bonigo and Brandán-Briones, 2012].

The following results generalized Theorems 1 and 2 respectively, giving necessary and sufficient conditions for the diagnosability of the global system.

Theorem 3. *Let $G_1, G_2, \ldots, G_n$ be n components modeled by LTSs, then*

$$\textbf{\textit{diag}}(G_1 \times G_2 \times \cdots \times G_n) \Downarrow \overbrace{\begin{array}{l} \textbf{\textit{diag}}(G_1 \times G_2^f \times \cdots \times G_n^f) \wedge \\ \textbf{\textit{diag}}(G_1^f \times G_2 \times \cdots \times G_n^f) \wedge \\ \vdots \\ \textbf{\textit{diag}}(G_1^f \times G_2^f \times \cdots \times G_n) \end{array}}$$

Theorem 4. *Let $G_1, G_2, \ldots, G_n$ be n components modeled by LTSs, then*

$$\underbrace{\begin{array}{l} \textbf{\textit{diag}}(G_1) \wedge \textbf{\textit{diag}}(G_1 \times G_2^f \times \cdots \times G_n^f) \wedge \\ \textbf{\textit{diag}}(G_2) \wedge \textbf{\textit{diag}}(G_1^f \times G_2 \times \cdots \times G_n^f) \wedge \\ \vdots \\ \textbf{\textit{diag}}(G_n) \wedge \textbf{\textit{diag}}(G_1^f \times G_2^f \times \cdots \times G_n) \end{array}} \Downarrow \textbf{\textit{diag}}(G_1 \times G_2 \times \cdots \times G_n)$$

Their proofs can be inferred directly from results that can be found in [Bonigo and Brandán-Briones, 2012].

When the faults can occur in every component and $G_1^f \times G_2^f \times \cdots \times G_n \neq G_1 \times G_2 \times \cdots \times G_n$, our approach shows important advantages, however in the cases where $G_1^f \times G_2^f \times \cdots \times G_n = G_1 \times G_2 \times \cdots \times G_n$, the whole product is analyzed and the computation time of our method is equal to the classic one. Nevertheless, when a diagnosability analysis is performed it is because it is known that several faults can occur in different components and it is more likely that $G_1^f \times G_2^f \times \cdots \times G_n$ is smaller than $G_1 \times G_2 \times \cdots \times G_n$.

Moreover, the diagnosability analysis of each component and $G_1^f \times G_2^f \times \cdots \times G_n$ can be tested in parallel, allowing parallel analysis of diagnosability.

4 The DADDY tool

In the previous section we try to minimize the information that components needs to share to be able to decide the diagnosability property of the whole system. We now present our tool, called DADDY (from Distributed Analysis for distributed Discrete sYstems). DADDY implements the method presented above and the classic one (where the synchronous product is computed before the diagnosability analysis is performed). The tool is written in Python and has GNU GPL v3 license. It uses a standard format (.aut) for the description of each component and it also allows to see a graphical representation of the system. It can be downloaded from [Bonigo, 2012].

The tool receives as inputs the components of the system. These inputs are assumed to be diagnosable, if not, an alert message is returned. If the specifications, meaning the non faulty components, are not given, systems G_j^f, for $j \neq i$, are computed following Algorithm 1. Hence G_j^f is synchronized with G_i, and its diagnosability is checked using the twin plant method from [Jiang *et al.*, 2000]. Also, time t_i of such computation is registered.

As soon as it is known that a component interacting with fault free versions of the other ones is non diagnosable, applying Theorem 3, a non diagnosable verdict is returned. Moreover, using the fact that it is a distributed computation,

System	Diagnosable	Our method	Classic method
$A \times B$	yes	0.0027251243	0.024051904
		0.0028400421	0.023932933
		0.0028848648	0.024003028
		0.0029160976	0.025793075
		0.0032229423	0.023809194
$C \times D$	no	0.0041198730	0.015272855
		0.0040440559	0.015629053
		0.0042178630	0.015436887
		0.0040760040	0.009753942
		0.0047080516	0.015598058

Figure 5: Diagnosis results in seconds unit

when we find a non diagnosable component, the computation of all others components can be stopped. So, the resulting time of such computation is $min(t_i)$ with $1 \leq i \leq n$.

On the other hand, if every component interacting with the fault free version of the other ones is diagnosable, using the assumption that every G_i is diagnosable by its own, we can conclude that $G_1 \times \cdots \times G_n$ is diagnosable applying Theorem 4. In this case, the diagnosability of every component is computed (in parallel) and the required time is $max(t_i)$ with $1 \leq i \leq n$.

We can see in table from Figure 5 that the diagnosability analysis results obtained by DADDY are consistent with the ones presented in our previous examples. We can also see that our method can be almost ten times faster than the classical one. If we consider systems n_1, n_2, n_3 from exaples/sample5 in [Bonigo, 2012], a non diagnosable result is obtained (as $n_1^f \times n_2 \times n_3^f$ is not diagnosable) in 0.16974902153 seconds with our method while the classical one does not reach a result after more than 24 hours. This shows an important improvement with respect to the classical method when the number of components grows.

5 Conclusions and Future Work

We have presented a new framework for the distributed diagnosability analysis of concurrent systems. We remove the assumption that a fault can only occur in a single component (which is usually made in distributed systems) and allow to analyze more general systems. The method presented in this paper parallelized the analysis leading, in general, to an important reduction in the computing time. The theoretical results are illustrated by several examples and supported by experimental results obtained with the DADDY tool.

We plan to continue trying to keep reducing the system in order to obtain minimal components from which we can infer the diagnosability of the original global system. In addition, we intend to relax the assumption that the communicating (synchronizing) events are observable.

Furthermore, even if the framework presented in this paper allows the distribution of the analysis, the formalism to model the systems is still sequential (product of LTSs) and can suffer of state space explosion making the twin plant method to check its diagnosability still prohibitive. We are working to extend such analysis to concurrent models such as Petri Nets.

References

[Baier and Katoen, 2008] C. Baier and J-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[Bonigo and Brandán-Briones, 2012] G. Bonigo and L. Brandán-Briones. *Trabajo Final para la Licenciatura en Ciencias de la Computación: Análisis de Diagnosticabilidad en Sistemas Distribuidos*. http://www.famaf.unc.edu.ar/institucional/biblioteca/trabajos/638/16627.pdf, 2012.

[Bonigo, 2012] G. Bonigo. Daddy. `https://code.google.com/p/daddy/`, 2012.

[Brandán-Briones and Madalinski, 2011] L. Brandán-Briones and A. Madalinski. Bounded predictability for faulty discrete event systems. In *SCCC*, pages 815–830, 2011.

[Brandán-Briones *et al.*, 2008] L. Brandán-Briones, A. Lazovik, and P. Dague. Optimizing the system observability level for diagnosability. In *ISoLA*, pages 815–830, 2008.

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

[Genc and Lafortune, 2003] S. Genc and S. Lafortune. Distributed diagnosis of discrete-event systems using petri nets. In *ICATPN*, pages 316–336, 2003.

[Jiang *et al.*, 2000] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46:1318–1321, 2000.

[Madalinski *et al.*, 2010] A. Madalinski, F. Nouioua, and P. Dague. Diagnosability verification with petri net unfoldings. *Int. J. Know.-Based Intell. Eng. Syst.*, 14(2):49–55, April 2010.

[Pencolé, 2004] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. In *ECAI*, pages 43–47, 2004.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[Schumann and Huang, 2008] A. Schumann and J. Huang. A scalable jointree algorithm for diagnosability. In *AAAI*, pages 535–540, 2008.

[Schumann and Pencolé, 2007] A. Schumann and Y. Pencolé. Scalable diagnosability checking of event-driven system. In *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI07*, pages 575–580, 2007.

[Ye and Dague, 2012] L. Ye and P. Dague. Diagnosability analysis for self-observed distributed discrete event systems. In *VALID*, pages 93–98, 2012.

Diagnosis of Discrete Event Systems by Independent Windows

Xingyu Su[1,2] and **Alban Grastien**[1,2]
[1]Optimisation Research Group, NICTA, Australia
[2]Artificial Intelligence Group, Australian National University, Australia
e-mail: u4383016@anu.edu.au, alban.grastien@nicta.com.au

Abstract

We propose new algorithms for diagnosis of discrete event systems that slice the observations into windows, each diagnosed independently. Doing so allows to cope with intermittent observations and to ignore the overhead of maintaining a precise estimate of the system state. We show how diagnosability can be asserted using this approach – and any diagnosis algorithm in general – through the notion of a *simulation*, which is a modified model that simulates how the diagnosis algorithm computes the diagnosis.

1 Introduction

Diagnosis of discrete event systems (DES) is performed by computing the paths on the complete model that generate the observations received on the system, or equivalently the belief state of system. This belief state can be computed on-line, by iteratively computing the set of states that can be reached from the current belief state through transitions that would produce exactly the next observation. If this is done explicitly [Baroni *et al.*, 1999], the number of these states makes the approach inapplicable for many real-world problems. Symbolic approaches have been proposed where the states are represented in propositional logic, e.g., Binary Decision Diagram (BDD), but this representation is also subject to exponential blow-up. Finally, the potential belief states can be pre-computed [Sampath *et al.*, 1995], but their number is double exponential this time.

In this last approach, the exponential blow-up stirs from the fact that the diagnoser maintains all the information it can about the observations it received so far. For instance, the current belief state could be $\mathcal{B}_1$ instead of $\mathcal{B}_2 \simeq \mathcal{B}_1$ because of an early observation.

In this paper, we propose several algorithms for computing diagnosis of DES, which share the property that they only consider the most recent observations. On top of the computational advantages mentioned above, these algorithms are able to deal with intermittent loss of communication, whereby the state of the system becomes unknown.

On the other hand, independent windows may potentially lead to imprecise diagnosis. In order to measure the quality of this approach, we adapt the well-known criterion of diagnosability. Diagnosability is the property of an observable system that states that, using the model, a fault can be diagnosed after it occurs. We extend this definition to algorithms, and say that an algorithm is diagnosable if it will diagnose the fault after it occurs. We show that diagnosability can be tested using known algorithms, on the condition that simulation can be built, i.e., a modified model that *simulates* the diagnostic algorithm decision.

This paper is organised as follows. Section 2 provides the traditional definitions of DES model and diagnosis. Section 3 defines the generic notion of diagnostic algorithm and the issue of proving that an algorithm is diagnosable. Section 4 presents the independent-window algorithms and how diagnosability can be tested for such algorithms. Section 5 demonstrates an example on how to implement diagnosability testing.

2 Diagnosis and Preliminaries

2.1 Diagnosis of DES

We choose DES as a modelling framework [Cassandras and Lafortune, 2008]. A DES models dynamic systems at a discrete level. We use automata to represent DES, although we also give definitions at the language level. Faults can be defined at the event level or the state level; to simplify the following definitions, we consider faulty states.

Definition 1 (Automaton) *An* automaton *is a tuple* $\langle Q, \Sigma, T, I, L\rangle$ *where* Q *is a finite set of* states, Σ *is a finite set of* events, $T \subseteq Q \times \Sigma \times Q$ *is a set of* transitions, $I \subseteq Q$ *is the set of* initial states, *and* $L : Q \to \{N, F\}$ *is a* mode label function *where* N *stands for* nominal *and* F *for* faulty *such that* $(\langle q, e, q'\rangle \in T \wedge L(q) = F) \Rightarrow L(q') = F$.

The system is modelled by an automaton with a set $\Sigma_o \subseteq \Sigma$ of observable events. For ease, we can also see the model as a pair of languages $\mathcal{L} = \mathcal{L}_N \cup \mathcal{L}_F$ such that $e_1, \ldots, e_k \in \mathcal{L}_l$ ($l \in \{N, F\}$) iff there exists a trace $q_0 \xrightarrow{e_1} \ldots \xrightarrow{e_k} q_k$ where $q_0 \in I$ and $L(q_k) = l$.

The observation $obs(\sigma)$ of $\sigma \in \Sigma\star$ is defined as the restriction of σ to the set of observable events:

$$obs(e_1, \ldots, e_k) = \begin{cases} \varepsilon & \text{if } k = 0 \\ e_1, obs(e_2, \ldots, e_k) & \text{if } e_1 \in \Sigma_o \\ obs(e_2, \ldots, e_k) & \text{if } e_1 \in \Sigma \setminus \Sigma_o \end{cases}$$

Fig. 1 shows a graphical representation for a DES model. It visualises a DES model and its states, initial

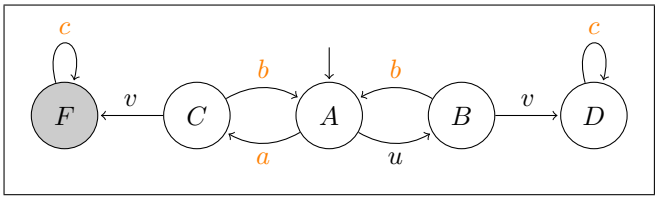


Figure 1: Example DES model

state, transitions, events and the results of the mode label function L. Finally, observable events are a, b, c, with the rest being unobservable, u and v.

A diagnosis problem is therefore defined as a model and an observation. In general, the diagnosis should indicate whether the system is in nominal mode, in faulty mode, or whether it cannot decide, i.e., ambiguous state. In this work, the diagnosis policy does not distinguish between nominal mode and ambiguous mode, i.e., the diagnoser assumes that the system is not faulty unless proved otherwise.

Definition 2 (Diagnosis) *The* diagnosis $\Delta(M, o)$ *of observation o using model M is defined by:*

- N *if* $\exists \sigma \in \mathcal{L}_N.\ obs(\sigma) = o$ *and*
- F *otherwise.*

Most existing algorithms implement a similar definition [Sampath *et al.*, 1995; Baroni *et al.*, 1999; Pencolé and Cordier, 2005; Su and Wonham, 2005; Grastien *et al.*, 2007; Kan John and Grastien, 2008]. Because her work is the original one, we shall call Δ the *Sampath* diagnosis.

Observation	Diagnoser	Diagnosis
a, b, b, b, b, c, c, c	Sampath	N
a, b, b, b, a, c, c, c	Sampath	F
b, a, b, a, c, c, c, c	Sampath	F

Table 1: Example of diagnosis results

Example Tab. 1 shows some examples of diagnosis results for the system on Figure 1. For instance, given a sequence of observations a, b, b, b, b, c, c, c, the Sampath Diagnoser will return a nominal diagnosis.

2.2 Diagnosability of DES

Diagnosability is the question whether a fault will always be diagnosed. This property is very desirable and the system designer might want to enforce it.

Diagnosability can be checked by searching for faulty system behaviours that cannot be diagnosed precisely (in dynamic systems where the aim is to diagnose the fault eventually, these system behaviours must be infinite). Any such example is a proof that the system is not diagnosable; it is called an ambiguous trace. Failure to find such a trace proves that the system is diagnosable, assuming the search was complete. In this reduction of the diagnosability problem, the model is used twice [Grastien and Torta, 2011]: it is used i) to find the faulty trace on the system and ii) to (unsuccessfully) diagnose the trace. However we may have to use different models for these two usages, one model for the system and the other model for the diagnosis. If the system model is abstracted before being used for diagnosis, the abstracted model should not be used to generate the faulty trace (that trace may not be a possible system behaviour); on the other hand, whether the trace can be diagnosed should be tested with the diagnosis model.

Definition 3 (Diagnosability of a model) *A diagnosis model M is* diagnosable *w.r.t a system model M' if the following holds true:*

$$\exists n \in \mathbf{N}.\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma\star\,.$$
$$(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow \Delta(M, obs(st)) = F)\,.$$

Given a faulty behaviour s, given an additional n events (represented by t), the diagnosis (using the model M) of the observation $obs(st)$ of st should be "faulty".

Jiang et al. [2001] have shown that diagnosability can be checked in polynomial time with respect to the number of states. To this end they build a structure called a *twin plant*, which is the classical automata synchronisation of two copies of the model on the observable events. M is diagnosable w.r.t M' iff the twin plant contains no infinite cycle of states $\langle q, q' \rangle$ where $L(q) = N$ and $L(q') = F$.

3 Diagnostic Algorithms

In this section we discuss the notion of diagnostic algorithm which is a possibly imprecise implementation of the diagnosis presented in Definition 2.

Definition 4 (Diagnostic Algorithm) *A* diagnostic algorithm *is a function* A : MODELS $\times\ \Sigma_o\star \rightarrow \{N, F\}$ *where* MODELS *is the set of possible models and the following holds:*

Monotonicity:
$$A(M, o) = F \Rightarrow \forall e \in \Sigma_o.\ A(M, o\ e) = F\text{; and}$$

Correctness: $A(M, o) = F \Rightarrow \Delta(M, o) = F$.

The first condition ensures that the diagnosis is monotonic [Lamperti and Zanella, 2007], i.e., that if a fault has been diagnosed, this conclusion will not be withdrawn. The second condition of the definition ensures that the diagnosis is correct, i.e., that the algorithm returns "faulty" only when the system is faulty (the converse may not hold).

The diagnosis problem is a complex task. We are interested in considering algorithms that may return results that are less precise than the Sampath diagnosis, but that run faster. We also want however to be able to give guarantees about the output of the algorithm, e.g., diagnosability.

Definition 5 (Diagnosability of an Algorithm) *The diagnostic algorithm A is* diagnosable *for a diagnosis model M w.r.t. a system model M' if the following holds true:*

$$\exists n \in \mathbf{N}.\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma\star\,.$$
$$(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow A(M, obs(st)) = F)\,.$$

This definition is very similar to that of diagnosability (Definition 3) except that we require the output of the algorithm A (and not the Sampath diagnosis itself) to be "faulty".

Diagnosability of a model can be tested using the twin plant approach. In general however, we do not know how to compute the diagnosability of a diagnostic algorithm. To this end we propose to define a new model $si(M, A)$ that "simulates" the behaviour of the diagnosis algorithm.

Definition 6 (Simulation) *Given a diagnostic algorithm A and a model M, the* simulation *of A and M is an automaton $si(M, A)$ s.t. $\forall o \in \Sigma_o \star . A(M, o) = \Delta(si(M, A), o)$.*

Using the simulation, we can use the following theorem to prove diagnosability.

Theorem 1 *Algorithm A is diagnosable for a diagnosis model M w.r.t. a system model M' iff $si(M, A)$ is diagnosable w.r.t. M' where $si(M, A)$ is the simulation of M and A.*

Proof of Theorem 1 Based on definition 5 for diagnosability and the condition that diagnosis of an algorithm A is equivalent to that of Sampath Diagnoser using a simulated model, $si(M, A)$, the diagnosability of Sampath Diagnoser using $si(M, A)$ implies the diagnosis on a trace st should return F if it reaches faulty state, i.e., $\Delta(obs(st)) = F$.

Hence, the diagnosis of the algorithm A on the trace st should also return F if it reaches faulty state. $A(M, obs(st)) = F$.

Therefore, it is proved that the diagnosability of (A, M, M') is equivalent to the diagnosability of $(\Delta, M, si(M, A))$. □

Theorem 1 shows that it is possible to use the twin plant method to decide whether algorithm A is diagnosable for M.

4 Independent-Windows Algorithm

A time window is defined as a slice of a sequence of observations. Independent-windows algorithms perform diagnosis on either a single window or a collection of time windows. In this section, we explain the motivations for the independent-windows algorithms, four different independent-window algorithms, and their associated simulation.

4.1 Motivations

There are essentially three main motivations for independent-windows algorithms: the flexibility of this approach, its potential impact on the diagnosis complexity, and its use to tackle the problem of masking.

The independent-windows algorithms perform independent diagnosis analyses on separate windows. Because each diagnosis is performed separately, we can skip the ones that we believe are not relevant. Consider for instance a problem occurring on a web-service whose activity is logged. It might be tedious to go through all the logs to produce the diagnosis; it is more likely that analysing the data some time before the incident was detected will be enough.

Second, we expect the complexity of diagnosis to become more manageable by considering independent-windows. Diagnosis of DES as defined above is about maintaining a belief state, and the number of such belief states is exponential in the number of system states (double exponential in the number of variables). Whichever way the belief state is computed (off-line [Sampath *et al.*, 1995] or on-line [Baroni *et al.*, 1999], even using decomposition or symbolic tools), diagnosis quickly becomes untrackable. With windows of fixed size however, the complexity changes completely. For instance, given b observable events in the model and a size k of the window, the number of Sampath belief states that can be reached is $\Sigma_{i\in\{0,\dots,k\}} b^i = \frac{b^{k+1}-1}{b-1} = O(b^k)$. Practically, this means that a Sampath diagnoser of polynomial size can be built, or the BDD unfolding on k transitions of the system model.

Finally when the communication layer used to transmit the observation is also subject to faults, certain observations are *masked* which means that the complete sequence of observations is not available. Independent-windows algorithms are less subject to this issue since they reset the diagnosis at the beginning of every window.

4.2 Single-Window Diagnosis

Single-window diagnosis computes the diagnosis on windows, which are small chunks of observations. Given a sequence $o = o_1, \dots, o_n \dots$ of observations, given two indexes $i < j$, the window $o[i, j]$ is defined as the subsequence of observations $o_i, \dots, o_j$. Notice that, if $j \leq n$, the length of the window (number of observations) is $j - i + 1$; if $i < n$, its size is 0; otherwise, the size is $n - i + 1$.

Given a window $o_1, \dots, o_k$, a system behaviour $\sigma \in \mathcal{L}$ is compatible with the window if the window appears in the observations of σ; formally if there exist two sequences of observations $o'_1, \dots, o'_m$ and $o''_1, \dots, o''_p$ such that $obs(\sigma) = o'_1, \dots, o'_m, o_1, \dots, o_k, o''_1, \dots, o''_p$.

Definition 7 (Single-Window Algorithm) *The* single-window algorithm *denoted $W_{i,j}$ is defined as follows:*

- *$W_{i,j}(M, o) = N$ if there exists a nominal behaviour $\sigma \in \mathcal{L}_N$ compatible with $o[i, j]$;*
- *$W_{i,j}(M, o) = F$ otherwise.*

Observe that the diagnosis $W_{i,j}(M, o)$ can be simply computed as the diagnosis $\Delta(M', o[i, j])$ where M' is the modified model M where all reachable states are made initial.

4.3 Windows-Based Diagnosis

Based on single-window diagnosis, windows-based diagnosis facilitates various ways to slice a sequence of observations into time windows, each of which comes with its strengths. The final output will be a conjunction of diagnostic results of several time windows, i.e., the system is in faulty mode if the diagnosis on at least one time window returns faulty; it is in nominal mode if all time windows return nominal.

Given an observation $o \in \Sigma_o{}^\star$, given the sub-words $o_1, \dots, o_n$ of o and Time-Windows $= \{[i_1, j_1], \dots\}$ where i_i and j_i are integer indexes as defined in Section 4.2, the following diagnosis is correct:

$$A(M, o) = \begin{cases} N & \text{if } \forall [i, j] \in \text{Time-Windows}. \\ & \quad W_{i,j}(M, o) = N \\ F & \text{otherwise.} \end{cases}$$

Notice that more observations implies that the observations of the windows increase, which increases the chances for a single-window diagnosis to return faulty, hence the monotonicity of the algorithm.

The definition in Section 4.3 facilitates future time windows. If the system has been diagnosed faulty, then the diagnosis of any following time windows will return faulty. Therefore, windows-based diagnosis obeys the properties of monotonicity and correctness as defined in Definition 4. In the next section, we will illustrate windows-based diagnosis by four examples. In Section 4.5, we will examine the diagnosability of windows-based diagnosis.

4.4 Examples of Windows-Based Diagnosis

We will provide descriptions for the time windows of four independent-windows algorithms and their expected benefits. These algorithms should then follow the definitions in Section 4.2 and 4.3.

Algorithm 1 (Al_1) Al_1 means slicing a sequence of observations every k observations:
Time-Windows-$Al_1 = \{[1,k],[k+1,2k],\dots\}$.

Algorithm 2 (Al_2) Al_2 makes sure the windows overlap so that observations which are consecutive appear in the same window:
Time-Windows-$Al_1 = \{[1,k],[k+1,2k],\dots\} \cup \{[k'+1,k'+k],[k'+k+1,k'+2k],\dots\}$ where $k' = \lfloor \frac{k}{2} \rfloor$.

Algorithm 3 (Al_3) Al_3 aims at slicing observations and running diagnosis at random times, yet frequently reoccurring, times:
Time-Windows-$Al_3 = \{[i_1, i_1+k],[i_2,i_2+k],\dots\}$ such that there exists a maximum delay d, and for all index x, $0 < i_x - i_{x-1} < d$.

Algorithm 4 (Al_4) Al_4 proposes sliding time windows to run diagnosis. A sliding time window moves along a observation flow by one observation at a time.
Time-Windows-$Al_4 = \{[1,k],[2,k+1],\dots\}$.

Observation	Slice	Diagnosis	Output
a,b,b,b,b,c,c,c	(a,b,b,b)	N	N
	(b,c,c,c)	N	
a,b,b,b,a,c,c,c	(a,b,b,b)	N	F
	(a,c,c,c)	F	
b,a,b,a,c,c,c,c	(b,a,b,a)	N	N
	(c,c,c,c)	N	

Table 2: Examples of Al_1

Example Tab. 2 shows the results of Al_1 running on the same set of observations as in Tab. 1. Based on the DES model in Fig. 1, given the first input: a,b,b,b,b,c,c,c, using Al_1 with the window size to be 4, we get 2 slices: (a,b,b,b) and (b,c,c,c). Running Al_1 on the first slice leads to N. Also, the diagnostic result of the second slice is N. Therefore, this sequence of observations is N. One of the slices of the second observation ends in F. Thus, Al_1 will return F. In the third observation, the diagnostic results of both slices are N. However, Tab. 1 shows this observation should be F. This demonstrates that Al_1 has its drawbacks of imprecise diagnosis in some situations. Algorithm Al_2 overcomes this issue.

Input	Slice	Diagnosis	Output
a,b,b,b,b,c,c,c	(a,b,b,b)	N	N
	(b,b,b,c)	N	
	(b,c,c,c)	N	
a,b,b,b,a,c,c,c	(a,b,b,b)	N	F
	(b,b,a,c)	F	
	(a,c,c,c)	F	
b,a,b,a,c,c,c,c	(b,a,b,a)	N	F
	(b,a,c,c)	F	
	(c,c,c,c)	N	

Table 3: Examples of Al_2

Tab. 3 is the result of running Al_2, which uses additional slices. We have seen that Al_1 cannot diagnose the fault in the third observation. In contrast, Al_2 will accurately return F.

4.5 Simulation of the system model M'

We now study diagnosability in order to evaluate windows-based algorithms by presenting a simulation of the window-based algorithms. Remember that, while the simulation generates a model with more states than the original one, the simulation is only used for diagnosability testing. Furthermore, the states and transitions of the automata do not need to be explicitly represented, but can be represented symbolically, e.g. BDD.

The simulation $si(M, Al_1)$ of the window-based algorithm Al_1 and the model M is an automaton that contains $k+1$ copies of the states of M, where each state $\langle q,i\rangle$ of $f(M,Al_1)$ records not only the system state q but also the number i of observations in the current window. When the number of observations has reached k, then the state of the simulation is "reset", i.e., a transition takes the simulation from state $\langle q,k\rangle$ to state $\langle q',0\rangle$ such that q' and q only have in common that their diagnostic information is the same: $L(q) = L(q')$.

Definition 8 *The k-simulation of $M = \langle Q,\Sigma,T,I,L\rangle$ is the automaton $M' = \langle Q',\Sigma',T',I',L'\rangle$ where $Q' = Q \times \{0,\dots,k\}$, $\Sigma' = \Sigma \cup \{\varepsilon\}$, $I' = I \times \{0\}$, $L'(\langle q,i\rangle) = L(q)$, and $T' = T_u \cup T_o \cup T_\varepsilon$ defined by:*

- $T_u = \{\langle\langle q,i\rangle,u,\langle q',i\rangle\rangle \mid i \in \{0,\dots,k\} \wedge \langle q,u,q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$,
- $T_o = \{\langle\langle q,i\rangle,o,\langle q',i+1\rangle\rangle \mid i \in \{0,\dots,k-1\} \wedge \langle q,o,q'\rangle \in T \wedge o \in \Sigma_o\}$, *and*
- $T_\varepsilon = \{\langle\langle q,k\rangle,\varepsilon,\langle q',0\rangle\rangle \mid L(q) = L(q')\}$.

Example Fig. 2 illustrates the Algorithm Al_1 simulation for the DES model in Fig. 1. F_0, F_1, F_2, F_3 and F_4 are faulty states, with the rest being nominal. Definition 8 says for every observable transition, create a link from one state to the next state on the next row,

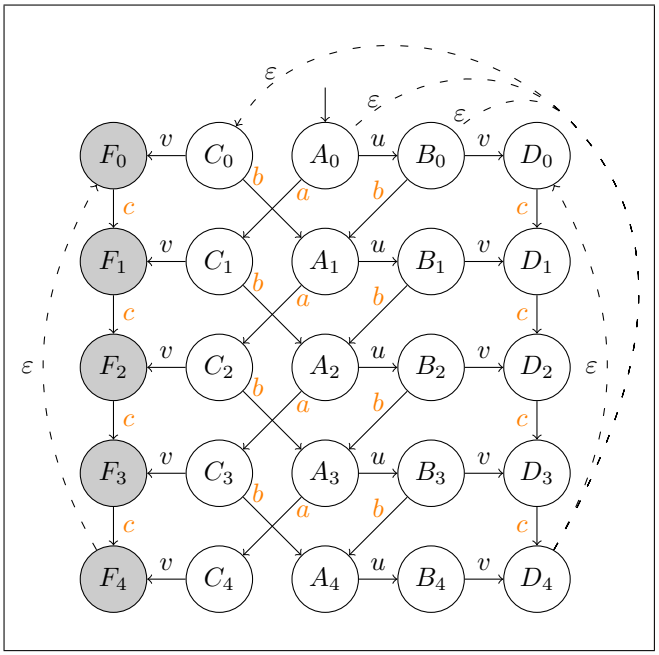


Figure 2: Part of Algorithm Al_1 simulation for the DES model in Fig. 1. Dotted lines also need to link A_4 to A_0, B_0, C_0 and D_0. Same applies to B_4 and C_4.

which represents the next time step; for every unobservable transition, create a link from one state to the next state within its row; for each faulty state at the end of one time window, create a link ε from F_4 to every faulty state at the beginning of one time window, i.e., F_0; for each nominal state at the end of the time window, create a link ε to every nominal state at the beginning of the time window.

There is one unique simulation that can be built for a system model M' and an algorithm A. Simulation is only used for diagnosability test and not for diagnosis. In the simulation of Al_2, Fig. 1 and 2 are synchronised on the initial states, i.e., A_0 and A_2, each leading to a simulation. For Al_3, every time window has a simulation, which may have different sizes as determined by the time windows of Al_3. Finally, those simulations will be synchronised on their initial states to reach the simulation of Al_3.

With the help k-simulation, we develop Theorem 2.

Theorem 2 *The k-simulation of a diagnosis model M as defined in Definition 8 is the simulation of Al_1 and M as defined in Definition 6.*

Proof of Theorem 2 By observing that a path on the k-simulation of M with $|o|$ observations is the concatenation of $\lceil |o|/k \rceil$ paths of length k such that the diagnostic information at the end of one such path is the same as the one at the end of the other path. Therefore all paths in the simulation consistent with the observations end in a faulty state iff all paths of one window end in a faulty state, which is the definition of the simulation of a diagnosis algorithm. □

The simulations, together with Theorem 1, allow to prove that Algorithm Al_1 with $k \geq 2$ is not diagnosable for the system of Figure 1; on the other hand, Algorithm Al_2 is diagnosable for any $k \geq 2$.

5 Experiments

We adopt BDD as the data structure for DES. We implement BDD using JDD, a Java library for creation and operations on BDD variables[1]. First of all, the input DES model is in *des_comp* format according to the Dia-Des project of Yannick Pencolé[2]. Second, we use BDD variables for every state, event, and then build transitions. Third, we synchronise multiple automata on shared events. Next, we build a twin plant and implement the Forward Algorithm of symbolic model checking in order to test diagnosability [Grastien, 2009]. This is the first approach to test diagnosability without introducing any windows-based algorithm. In the second approach, before building a twin plant, make a copy of the synchronised automaton, unfold that copy in order to get the simulation for the windows-based algorithm, and then build a twin plant to test diagnosability. All data and benchmark are available on request.

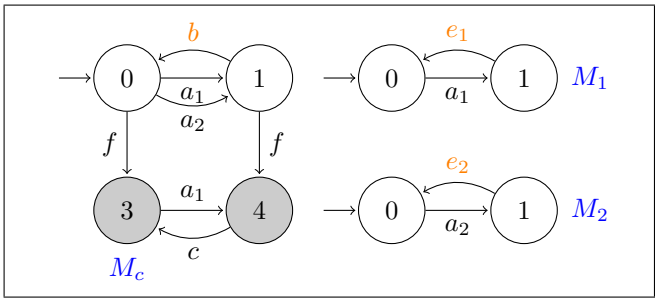


Figure 3: Experiments.

In order to test diagnosability when the windows-based algorithm is introduced, we build an example of factory operations, which consists of a few DES models. Fig. 3 shows a central model, M_c, and multiple operation plants, i.e., M_1, M_2, etc. M_c is to dispatch a job (a_i) to each plant (M_i) and receive feedback (e_i) once it is done. States 3 and 4 are shaded as faulty, with the rest being nominal. In the individual operation plants, e_i will be emitted. However, b will be emitted from M_c. Although c is not observable from M_c, only e_1 will be observed from M_1 if M_c enters a faulty scenario. To sum up, b and e_i are observable, while a_i, f and c are unobservable.

The experiments were run on a Ubuntu 12.04 computer with Intel Core i7-3610QM processor @3.3GHz and the total memory is 8 GB.

The results show that they are all diagnosable. Fig. 4 is a line chart to show log_{10} of the running time of diagnosability tests in milliseconds. The X-axis lists the number of model components to be synchronised. We start from 2 components, M_c and M_1, and then synchronise with M_2, M_3, etc. The Y-axis shows the time consumption in milliseconds. In Fig. 4, the filled boxes represent the time of running diagnosability test without any windows-based algorithm. The filled diamonds show the time when including the unfolding operation on the simulation of the windows-based algorithm as defined in Definition 8.

[1] javaddlib.sourceforge.net/jdd/

[2] homepages.laas.fr/ypencole/diades/html/index.html

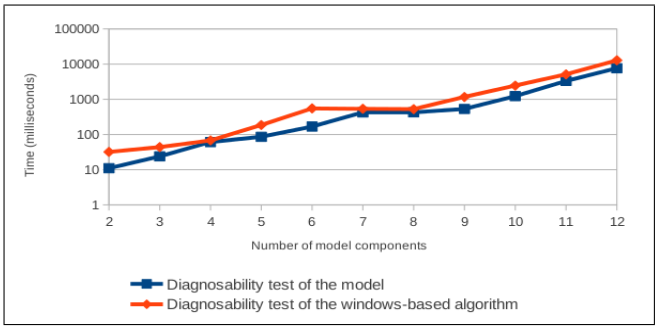


Figure 4: Running time

In summary, the trend of time consumption meets our expectation as the amount of model components grows. Also, diagnosability test on the windows-based algorithm takes more computation time.

6 Conclusion

We develop Independent-Windows Algorithms to diagnose DES and we study their diagnosability. We aim to slice a sequence of observations into time windows and diagnose them independently. This approach has the advantage to handle intermittent observations and does not require maintaining a precise estimate of the system state.

We begin with studying three key properties of a general diagnosis algorithm, such as monotonicity, correctness and diagnosability. We then prove Theorem 1 to determine the diagnosability of any diagnosis algorithm using a modified model and the twin plant method. This theorem enables us to compute the diagnosability of the new Independent-Windows Algorithms, so that we can measure how precise the diagnosis will be.

As we are certain that we will be able to determine the diagnosability of our new diagnosis algorithms, we are further motivated by the flexibility of independent windows, reduction of computation complexity and avoidance of the masking issues which may occur in transmission of observations. We then demonstrate four Independent-Windows Algorithms. Using the same set of observations, we compare the accuracy of Sampath Diagnoser and the new algorithms Al_1 and Al_2. This allows us to analyse the strengths of each diagnosis algorithms.

In order to determine the diagnosability of the Independent-Windows Algorithms as specified in Theorem 1, we show how to modify a DES model and simulate a diagnosis algorithm. After this, we use the output simulation to test the diagnosability of the new algorithms. Theorem 2 proves the correctness of generating the simulation. Finally, our experiments conclude that diagnosability testing on the windows-based algorithms can be implemented by BDD.

As an outline for our future work, we plan to study how prompt the windows-based diagnosis is. We will measure this by how fast it can reach a diagnostic result. We also want to examine the root cause of ambiguity if the diagnosis result is inability to decide whether the system is nominal or faulty. One possibility is to identify key states in the simulation, categorise them and retain this information.

Acknowledgements

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.

The authors want to thank Yannick Pencolé and the three anonymous reviewers for their useful comments.

References

[Baroni *et al.*, 1999] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence (AIJ)*, 110(1):135–183, 1999.

[Cassandras and Lafortune, 2008] Ch. Cassandras and S. Lafortune. *Introduction to discrete event systems (2nd ed.)*. Kluwer Academic Publishers, 2008.

[Grastien and Torta, 2011] A. Grastien and G. Torta. A theory of abstraction for diagnosis of dicrete-event systems. In *Ninth Symposium on Abstraction, Reformulation and Approximation (SARA-11)*, pages 50–57, 2011.

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, pages 305–310, 2007.

[Grastien, 2009] A. Grastien. Symbolic testing of diagnosability. In *20th International Workshop on Principles of Diagnosis (DX-09)*, pages 131–138, 2009.

[Jiang *et al.*, 2001] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 46(8):1318–1321, 2001.

[Kan John and Grastien, 2008] P. Kan John and A. Grastien. Local consistency and junction tree for diagnosis of discrete-event systems. In *Eighteenth European Conference on Artificial Intelligence (ECAI-08)*, 2008.

[Lamperti and Zanella, 2007] G. Lamperti and M. Zanella. On monotonic monitoring of discrete-event systems. In *Eighteenth International Workshop on Principles of Diagnosis (DX-07)*, pages 130–137, 2007.

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

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 40(9):1555–1575, 1995.

[Su and Wonham, 2005] R. Su and W. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 50(12):1923–1935, 2005.

Minimal Diagnosis of Discrete-Event Systems*

Xiangfu Zhao[1], **Gianfranco Lamperti**[2], and **Dantong Ouyang**[3]
[1]Department of Computer Science, Zhejiang Normal University, Jinhua, China
[2]Department of Information Engineering, University of Brescia, Brescia, Italy
[3]School of Computer Science and Technology, Jilin University, Changchun, China
Email: xiangfuzhao@gmail.com, gianfranco.lamperti@ing.unibs.it, ouyd@jlu.edu.cn

Abstract

Model-based diagnosis of discrete-event systems (DESs) is an active research topic. In most previous works, *all* candidate diagnoses are generated, which is bound to be hugely complex, especially when a large number of non-minimal candidates are involved. However, according to *Occam's Razor*, only minimal diagnoses are most valuable. In order to cut down the complexity and to get more likely candidates, a sort of minimal diagnosis for DESs is proposed. Then, a corresponding minimal diagnoser is defined for online generation of minimal diagnoses. Furthermore, two sorts of minimal diagnosability of DESs are presented, to either strongly or weakly determine whether a minimal diagnosis has definitely occurred or not. Finally, relevant necessary and sufficient conditions for testing minimal diagnosability of DESs are analyzed.

1 Introduction

Coping with major disasters, such as the crashed space shuttle in Soviet Russia in 1967, the nuclear accident in Fukushima (Japan) in 2011, or the large-scale blackout in India in 2012, is a challenge for preserving the safety of society. Model-based diagnosis can be a useful tool in this direction, especially when new high-tech devices are involved, as it does not rely on experts' experience, which is scarcely obtained for such devices.

Since the seminal work [Sampath *et al.*, 1995] on diagnosis of discrete-event systems (DESs) [Cassandras and Lafortune, 2008], where the notion of *diagnoser* is proposed, model-based diagnosis of DESs has attracted increasing attention. In order to deal with the complexity of a centralized diagnoser based on a global system model, decentralized diagnosers based on decentralized models are proposed in [Pencolé and Cordier, 2005]. Recently, fuzzy information or stochastic events are injected into DESs, and the corresponding fuzzy diagnosers [Liu and Qiu, 2009] or stochastic diagnosers [Thorsley and Teneketzis, 2005] are proposed. Besides diagnoser-based approaches, a history-based approach [Lamperti and Zanella, 2003], and a consistency-based approach [Grastien *et al.*, 2012] to diagnosis of DESs, are also proposed.

However, to our knowledge, in most previous approaches for diagnosis of DESs, *all* possible candidate diagnoses are generated, even when there are set-containment relationships among them, in other words, when non-minimal diagnoses are included. For example, in candidate diagnoses $\{f_1\}$,$\{f_1, f_2\}$,$\{f_1, f_3\}$,$\{f_1, f_2, f_3\}$, only $\{f_1\}$ is minimal. According to *Occam's Razor* and the principle of joint-probability distribution (assuming that the failure possibility is the same for each fault of the system in general), only set-containment minimal diagnoses are more probable and, as such, more valuable. Therefore, determining just *minimal* diagnoses of DESs is an interesting problem, just as it is in static systems [Reiter, 1987; de Kleer and Williams, 1987]. Also complexity is reduced considerably, as non-minimal diagnoses are not considered. In order to (online) diagnose a DES, a *minimal diagnoser* is defined to store only minimal diagnoses. In addition, two novel notions of *minimal diagnosability* are defined and tested by a minimal diagnoser.

2 Language and FSM for Diagnosis of DESs

A DES is a (deterministic) finite state machine (FSM) G = (Q, Σ, T, q_0), where:

- Q is the set of states;
- Σ is the set of events, including observable events (Σ_o) and unobservable events (Σ_{uo}), with $\Sigma = \Sigma_o \uplus \Sigma_{uo}$ (with $\uplus$ denoting the union of two disjoint sets). $\Sigma_f = \{f_1, f_2, \cdots, f_m\}$, $\Sigma_f \subseteq \Sigma_{uo}$, is the set of failure events to be inferred.
- $T \subseteq Q \times \Sigma \times Q$, is the set of transitions like (q_i, e, q_j) (or $q_i \xrightarrow{e} q_j$ or $T(q_i, e) = q_j$) from state q_i to state q_j when event e is on state q_i.
- $q_0 \in Q$, is the initial state of the system.

Given an FSM G, all possible *traces* (behavior) generated from q_0 to a certain state in G, are described as a prefix-closed language $L(G)$, shorted by L, and $L \subseteq \Sigma^*$. We assume that L is *live* for simplicity, that is, there is at least one transition exiting each state, and there is no cycle of unobservable events [Sampath *et al.*, 1995]. Symbol ε denotes the empty trace. We extend the notion of a transition event to a *string* of transition events as follows:

- $q \xrightarrow{\varepsilon} q$ always holds;
- For $s \in \Sigma^*$ and $\sigma \in \Sigma$, $q \xrightarrow{s\sigma} q'$ holds whenever $q \xrightarrow{s} q''$ and $q'' \xrightarrow{\sigma} q'$ hold for $q'' \in Q$.

Let $q \xrightarrow{s}$ denote that, for $s \in \Sigma^*$, there exists at least a state $q' \in Q$, such that $q \xrightarrow{s} q'$ holds. $L/s = \{t \in \Sigma^* \mid st \in L\}$, is the post-language of L after string $s \in L$.

$Prj_{\Sigma_o}: \Sigma^* \to \Sigma_o^*$, denotes how a trace is mapped onto an observable sequence:

*This work was supported in part by NSFC under Grant Nos. 61003101 and 61272208; Zhejiang Provincial Natural Science Foundation under Grant No. Y1100191.

- $Prj_{\Sigma_o}(\varepsilon) = \varepsilon$;
- $Prj_{\Sigma_o}(\sigma) = \sigma$ if $\sigma \in \Sigma_o$;
- $Prj_{\Sigma_o}(\sigma) = \varepsilon$ if $\sigma \in \Sigma_{uo}$;
- $Prj_{\Sigma_o}(s\sigma) = Prj_{\Sigma_o}(s)Prj_{\Sigma_o}(\sigma)$, where $s \in \Sigma^*$, and $\sigma \in \Sigma$.

Conversely, $Prj^{-1}_{\Sigma_o}(y) = \{s \in L | Prj_{\Sigma_o}(s) = y\}$.

P_{Σ_f}: $\Sigma^* \to 2^{\Sigma_f}$ denotes the way to collect a set of failure events in a trace $s \in \Sigma^*$:

$$P_{\Sigma_f}(s) = \begin{cases} \emptyset & \text{if } s = \varepsilon; \\ \{f_i \mid f_i \in s\} & \text{otherwise.} \end{cases}$$

For example, $P_{\Sigma_f}(af_1bcf_2) = \{f_1, f_2\}$, where $a, b, c \in \Sigma - \Sigma_f$.

s_f denotes the final event of a non-empty trace $s \in \Sigma^+$.

$s_\delta = \{s \mid s \in L, s_f \in \delta, \delta \subseteq \Sigma_f, P_{\Sigma_f}(s) = \delta\}$ denotes the set of traces ending with one failure event of δ, and containing exactly the failure events of δ.

Let $\Delta = 2^{\Sigma_f}$ be all possible fault labels. We use N to denote the normal state with the empty fault set $\emptyset$.

Let $L(G, q)$ denote all traces in G originating in state q. $L_o(G, q) = \{s \mid s \in L(G, q), s = u\sigma, u \in \Sigma^*_{uo}, \sigma \in \Sigma_o\}$ denotes all the traces from q up to the *first observable* event, and $L_\sigma(G, q) = \{s \mid s \in L_o(G, q), s_f = \sigma\}$ denotes all the traces from q up to the first observable event σ.

A new (non-deterministic in general) FSM G^o = (Q^o, Σ_o, T^o, q_0) is defined as follows (with $L(G^o) = \{t \mid t = Prj_{\Sigma_o}(s), s \in L\}$; roughly speaking, each transition in G via an unobservable event is removed):

- Q^o = { q_0 } ∪ { $q \mid q \in Q$ has an observable event into it } denotes q_0 along with all observable states;
- $T^o \subseteq Q^o \times \Sigma_o \times Q^o$ denotes the set of transitions, each of which is from $q^o \in Q^o$ via subsequently the first observable event σ to an observable state $q^{o\prime}$: (q^o, σ, $q^{o\prime}$) $\in T^o$ if $T(q^o, s) = q^{o\prime}$, where $s \in L_\sigma(G, q^o)$.

In order to introduce the notion of a minimal diagnoser, we define a revised diagnoser G^d, based on the classical diagnoser G_d [Sampath *et al.*, 1995].

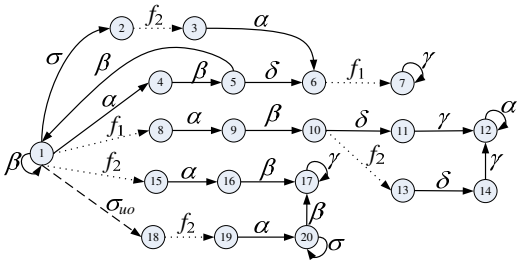
A revised diagnoser G^d for a DES G is defined as a deterministic FSM G^d = (Q^d, Σ_o, T^d, q^d_0), where:

- $q^d_0 = \{(q_0, N)\}$, assuming that G is normal to begin with. Any state $q^d \in Q^d$ is reachable from q^d_0 under transitions in T^d, written as $q^d = \{(q^o_1, l_1), \ldots, (q^o_n, l_n)\}$, where $q^o_i \in Q^o$, and $l_i \in \Delta$ (that is, l_i is of the form N, or a non-empty subset of Σ_f). In the following, we use the empty set $\emptyset$ to replace N for set computation. Let $Q^{od} = 2^{Q^o \times \Delta}$. Hence, $Q^d \subseteq Q^{od}$.
- The fault label propagation function, $LP : Q^o \times \Delta \times \Sigma^* \to \Delta$, is so defined. Given $q^o \in Q^o, l \in \Delta$, and $s \in L_o(G, q^o)$, LP propagates label l over string s from q^o as follows:

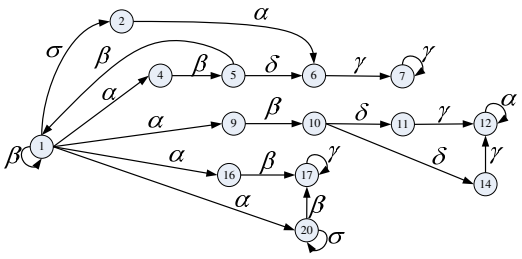
  $$LP(q^o,\ l,\ s) = \{f_i \mid f_i \in l \vee f_i \in s\}.$$

  Then, the generated state by transition function $T^d(q^d, \sigma)$ is defined as:

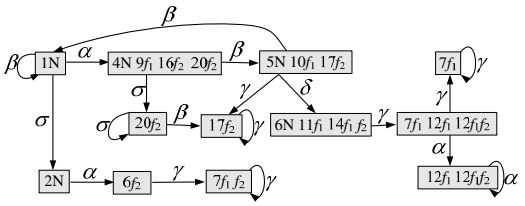
  $$\bigcup_{(q^o,\ l) \in q^d} \left(\bigcup_{s \in L_\sigma(G,\ q^o)} \{(T(q^o,\ s), LP(q^o, l, s))\} \right).$$



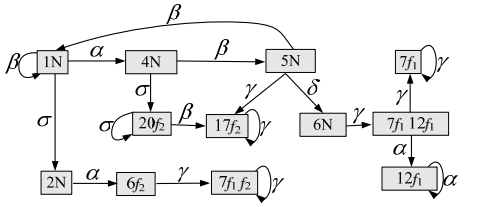
(a) DES model G.



(b) Non-deterministic FSM G^o for G.



(c) Revised diagnoser G^d for G.



(d) Minimal diagnoser G^{md} for G.

Figure 1: A DES and its two different diagnosers.

In other words, given the current state q^d_1 of revised diagnoser G^d, and the next observable event σ, the new state q^d_2 of G^d is generated as follows:

(1) For each (q^o, l) $\in q^d_1$, compute the reachable states of G by observation σ:

$$S(q^o, \sigma) = \{T(q^o, u\sigma) \mid u \in \Sigma^*_{uo}, \text{ and } \sigma \in \Sigma_o\}.$$

(2) Let $q^{o\prime} \in S(q^o, \sigma)$ with $T(q^o, u\sigma) = q^{o\prime}$, propagate label l associated with q^o to label l' associated with $q^{o\prime}$ as: $l' = l \cup \{f_i \mid f_i \in u\}$.

(3) Let q^d_2 be the set of all pairs ($q^{o\prime}$, l') generated by (1) and (2) above, for each $(q^o, l) \in q^d_1$.

Example 1. Consider the DES model shown in Fig. 1(a), where α, β, γ, δ, and σ are all observable events, f_1, f_2, and f_3 are all failure events, and σ_{uo} is an unobservable and non-failure event. Fig. 1(b) illustrates the G^o for DES G. The relevant revised diagnoser G^d is outlined in Fig. 1(c). (In figures, pairs (x, l) are denoted as xl for simplicity). ◇

Based on Fig. $1(c)$, we can see that all possible failure information is kept in G^d, which is exploited for minimization of fault sets in a minimal diagnoser.

In addition, to determine whether a fault has occurred definitely or not, the classical notion of diagnosability given in [Sampath *et al.*, 1995], is rephrased as follows (disregarding classification (types) of faults for a simpler representation):

Definition 1. *A prefix-closed and live language L is said to be diagnosable if, for any fault $f_i \in \Sigma_f$, we have:*

$$\exists n(n \in \mathbb{N})\ \forall s \in L(s_f = f_i)\ \forall t(t \in L/s)\ (\|t\| \geq n \Rightarrow D)$$

where the diagnosability condition D is defined as follows:

$$\omega \in Prj^{-1}_{\Sigma_o}(Prj_{\Sigma_o}(st)) \Rightarrow f_i \in \omega.$$

In other words, if a DES is diagnosable, then any failure event will be definitely recognized after its appearance, as long as the observation sequence is long enough.

Example 2. According to Definition 1, DES G in Fig. $1(a)$ is not diagnosable, as for observation sequences like $\alpha\beta\delta\gamma\alpha^k$, $k \in \mathbb{N}$, we cannot determine whether failure event f_2 has occurred or not. $\diamond$

3 Minimal Diagnosis of DESs

In this section, we are mainly concerned with minimal diagnosis. To this end, we first provide relevant definitions.

Definition 2. *Given a DES $G = (Q, \Sigma, T, q_0)$, with L being the corresponding language, and the current observation sequence $obs \in \Sigma_o^*$, $\delta \subseteq \Sigma_f$ is called a* candidate diagnosis *for obs (written as $\delta \models obs$), if there exists a string $s \in L$ with $s_f \in \Sigma_o$, such that $P_{\Sigma_f}(s) = \delta \wedge Prj_{\Sigma_o}(s) = obs$.*

That is, a candidate diagnosis is a set of failure events in a trace, whose projection on observations is just the same as the current observation sequence obs. Note that here $s_f \in \Sigma_o$ is required to be satisfied, since in general we use just the currently received observations, *after* the system failed to work normally, to infer a diagnosis to explain obs.

There may exist several strings in L, with each string having projection on observations equal to obs. Hence, there may exist several candidate diagnoses from different strings. The next definition formalizes the notion of minimal diagnosis by set-containment.

Definition 3. *Given a DES $G = (Q, \Sigma, T, q_0)$ and observation $obs \in \Sigma_o^*$. For any two candidate diagnoses δ_1, $\delta_2 \subseteq \Sigma_f$, $\delta_1 \models obs \wedge \delta_2 \models obs$, we write:*

- $\delta_1 \preceq \delta_2$ *if* $\delta_1 \subseteq \delta_2$*;*
- $\delta_1 \prec \delta_2$ *if* $\delta_1 \subset \delta_2$*;*
- $\delta_1 \prec\succ \delta_2$ *if* $\delta_1 \not\subseteq \delta_2 \wedge \delta_2 \not\subseteq \delta_1$.

$\delta \subseteq \Sigma_f$ is a minimal diagnosis *for obs, written as $\delta \models_{min} obs$, if $\delta \models obs$ and $\nexists \delta' \subseteq \Sigma_f$ such that $\delta' \models obs \wedge \delta' \prec \delta$. $\Delta_{min} = \{\delta \mid \delta \models_{min} obs\}$ is the set of minimal candidate diagnoses for obs.*

In other words, if δ is a minimal diagnosis, then any of its proper subset is not a diagnosis. The lesser the number of failure events of a candidate diagnosis δ is, the bigger the probability of δ will be (according to the principle of joint-probability distribution, assuming that each failure event has the same faulty probability).

4 Minimal Diagnosers for DESs

In order to derive all minimal diagnoses, we introduce the notion of a *minimal diagnoser*.

Definition 4. *Given a DES $G = (Q, \Sigma, T, q_0)$ and its revised diagnoser $G^d = (Q^d, \Sigma_o, T^d, q_0^d)$, a* minimal diagnoser *is an FSM $G^{md} = (Q^{md}, \Sigma_o, T^{md}, q_0^{md})$, where:*

(1) *For each $q_i^d \in Q^d$, there exists a corresponding minimized state $q_i^{md} \in Q^{md}$, obtained as follows: initially, $q_i^{md} = q_i^d$, then for each $(q^o, l) \in q_i^d$, any other $(q^{o\prime}, l') \in q_i^d$ with $l \prec l'$ will be removed from q_i^{md} (note: $q^{o\prime}$ may equal q^o), i.e., all the pairs with non-minimal labels will be deleted.*

(2) *For each transition $(q_i^d \xrightarrow{\sigma} q_j^d) \in T^d$ (where $\sigma \in \Sigma_o$, and $q_i^d, q_j^d \in Q^d$), there exists a corresponding transition $(q_i^{md} \xrightarrow{\sigma} q_j^{md}) \in T^{md}$ (where $q_i^{md}, q_j^{md} \in Q^{md}$).*

(3) *All states and transitions in G^{md} are yielded by* (1) *and* (2).

In other words, minimal diagnoser G^{md} (isomorphic to G^d) is a deterministic FSM with each state being usually more simplified than the corresponding state in G^d, since only minimal failure labels are kept in minimal diagnoses.

Based on Definition 4, some relevant properties of minimal diagnoser G^{md} are given:

(P_1) Let $q_i^{md} \in Q^{md}$; for each $(q_i^o, l_i) \in q_i^{md}$, there exists a state $q_i^d \in Q^d$ in G^d, such that $(q_i^o, l_i) \in q_i^d$.

(P_2) Let $q^{md} \in Q^{md}$; then, $(q^o, l), (q^{o\prime}, l') \in q^{md}$ iff there exist s, $s' \in L$ such that s_f, $s'_f \in \Sigma_o$, $T(q_0, s) = q^o$, $T(q_0, s') = q^{o\prime}$, $Prj_{\Sigma_o}(s) = Prj_{\Sigma_o}(s')$, $P_{\Sigma_f}(s) = l$, and $P_{\Sigma_f}(s') = l'$.

(P_3) Let $q^{md} \in Q^{md}$; if there exist $(q^o, l), (q^{o\prime}, l') \in q^{md}$, then $q^{o\prime}$ may be the same as q^o, i.e., the system may reach the same observable state q^o but with different minimal faults ($l \neq l'$).

(P_4) For each $q^{md} \in Q^{md}$ and for each $(q^o, l), (q^{o\prime}, l') \in q^{md}$, we have:

- $l = l' \Leftrightarrow l \subseteq l'$;
- $l \neq l' \Leftrightarrow l \prec\succ l'$, correspondingly.

(Note: here $q^{o\prime}$ can possibly equal q^o).

(P_5) Let $(q_i^{md} \xrightarrow{\sigma} q_j^{md}) \in T^{md}$; then for each $(q_j^o, l_j) \in q_j^{md}$, there exists $(q_i^o, l_i) \in q_i^{md}$ such that $l_i \subseteq l_j$.

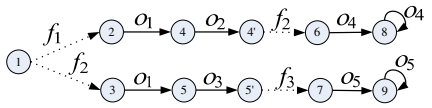
After (off-line) building minimal diagnoser G^{md} for DES G, and given the current observation obs, we can (online) synchronize obs with G^{md} to find the corresponding state in G^{md}, to directly get the minimal diagnoses.

Example 3. Considering the DES in Fig. $1(a)$, we easily obtain the current minimal diagnoses N (no fault by $(6, N)$) from G^{md} in Fig. $1(d)$, assuming $obs = \alpha\beta\delta$. Besides, based on Fig. $1(d)$, we get the new set of minimal diagnoses $\{f_1\}$ when we receive the additional observation γ. $\diamond$

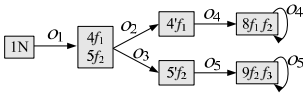
5 Strong Minimal Diagnosability of DESs

Before introducing the definition of minimal diagnosability of a DES G, to determine whether a fault set is diagnosable, we define all possible sets of faulty events L_f of G (with behavior L) as follows:

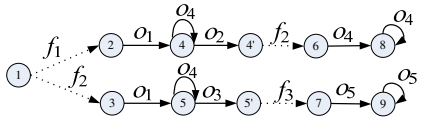
$$L_f = \bigcup_{s \in L \wedge s_f \in \Sigma_o} \{\{f_i \mid f_i \in s\}\}.$$



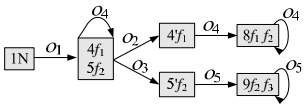
(a) DES model G_1.



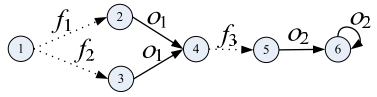
(a') Minimal diagnoser G_1^{md} for G_1.



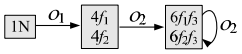
(b) DES model G_2.



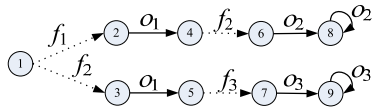
(b') Minimal diagnoser G_2^{md} for G_2.



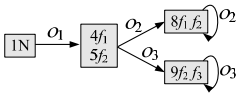
(c) DES model G_3.



(c') Minimal diagnoser G_3^{md} for G_3.



(d) DES model G_4.



(d') Minimal diagnoser G_4^{md} for G_4.

Figure 2: DES models and minimal diagnosers.

Definition 5. *A prefix-closed and live language L is said to be* strongly minimally diagnosable *if, for any fault set $\delta \in L_f$ and for any string $s \in s_\delta$, the following holds:*

(i) $\forall\, t\ (t \in L/s,\ t_f \in \Sigma_o,\ P_{\Sigma_f}(t) \subseteq \delta)\ \ \exists\, t'\ (t' \in L/(st),\ (tt')_f \in \Sigma_o,\ P_{\Sigma_f}(t') \subseteq \delta)\ \ ((\delta \models_{min} Prj_{\Sigma_o}(st)) \Rightarrow D_m^1)$;

(ii) $\exists\, n\ (n \in \mathbb{N})\ \ \forall\, t\ (t \in L/s,\ t_f \in \Sigma_o,\ P_{\Sigma_f}(t) \subseteq \delta)\ \ (\|t\| \geq n \Rightarrow ((\delta \models_{min} Prj_{\Sigma_o}(st)) \Rightarrow D_m^2))$;

where the strong minimal diagnosability conditions D_m^1 and D_m^2 are defined as follows:

D_m^1 : $(\omega \in Prj_{\Sigma_o}^{-1}(Prj_{\Sigma_o}(stt')) \Rightarrow (\delta \preceq P_{\Sigma_f}(\omega))$;

D_m^2 : $(\omega \in Prj_{\Sigma_o}^{-1}(Prj_{\Sigma_o}(st)) \Rightarrow (\delta \preceq P_{\Sigma_f}(\omega))$.

In other words, let s be a trace generated by G ending with a failure event set δ:

(i) For any continuation t of s without new fault events, the DES will always reach an observable state after a continuation t' of t (i.e., $(tt')_f \in \Sigma_o$) without new fault events, such that if δ is one minimal diagnosis for st then δ will be the *unique* minimal diagnosis for any trace with the same observation sequence in stt'.

(ii) Meanwhile, it is further required that there always exists a natural number n, such that, when the length of any continuation t of s is not less than n, if δ is the minimal diagnosis of st, then δ will be the *unique* minimal diagnosis for any trace with the same observation sequence in st.

Note: In contrast with the classical notion of diagnosability (Definition 1), here new conditions $P_{\Sigma_f}(t) \subseteq \delta$ and $P_{\Sigma_f}(t') \subseteq \delta$ are inserted, to restrict later subsequences, after the complete occurrence of δ, so as not to contain new failure events except those in δ, to make sure that δ is still kept as a candidate diagnosis.

The classical notion of diagnosability is only for any single failure event f_i (Definition 1), whereas our notion of minimal diagnosability of DESs is for a set δ of failure events, which must be *minimal* (compared to other possible candidates). Of course, both are required that any failure event f_i or any minimal fault set δ must be detected *definitely*, after their occurrences (within a finite delay), respectively.

However, there is no logical entailment between classical diagnosability and strong minimal diagnosability.

Example 4. DES G in Fig. $1(a)$ is strongly minimally diagnosable, yet not diagnosable, according to their definitions. Besides, DES G_4 in Fig. $2(d)$ is diagnosable, yet not strongly minimally diagnosable. $\diamond$

In order to present the necessary and sufficient conditions for strong minimal diagnosability of DESs, we first provide some related definitions and lemmas.

Definition 6. *A state $q^{md} \in Q^{md}$ is said to be* δ-certain*, if for any two pairs $(q^o, l), (q^{o'}, l') \in q^{md}$ ($q^{o'}$ can possibly equal q^o), we always have $l' = l$.*

A state $q^{md} \in Q^{md}$ is said to be δ-incomparable *if there exist two pairs (q^o, l), $(q^{o'}, l') \in q^{md}$ ($q^{o'}$ can possibly equal q^o) such that $l \prec\!\succ l'$.*

A state $q^{md} \in Q^{md}$ is said to be ambiguous *if there exist two pairs (q^o, l), $(q^o, l') \in q^{md}$ such that $l \prec\!\succ l'$.*

From Definition 6, we can also see that if a state q^{md} is ambiguous then it must be δ-incomparable too.

The following lemma applies to the three kinds of states.

Lemma 1. *For a minimal diagnoser G^{md} of DES G, the following properties hold.*

(i) *Let $T^{md}(q_0^{md}, s) = q^{md}$, $s \in \Sigma_o^*$. If state q^{md} (with $(q^o, l) \in q^{md}$) is δ-certain then for each $\omega \in Prj_{\Sigma_o}^{-1}(s)$ we have $l \preceq P_{\Sigma_f}(\omega)$.*

(ii) *If a state $q^{md} \in Q^{md}$ is δ-incomparable then for any two pairs $(q^o, l), (q^{o'}, l') \in q^{md}$ there exist two strings s, $s' \in L$ and s_f, $s'_f \in \Sigma_o$, such that $Prj_{\Sigma_o}(s) = Prj_{\Sigma_o}(s')$, $T^{md}(q_0^{md}, Prj_{\Sigma_o}(s)) = q^{md}$, $l = P_{\Sigma_f}(s)$, $l' = P_{\Sigma_f}(s')$, and $l \prec\!\succ l'$.*

(iii) *If a state $q^{md} \in Q^{md}$ is ambiguous then there exist two pairs $(q^o, l), (q^o, l') \in q^{md}$, two strings s, $s' \in L$, and s_f, $s'_f \in \Sigma_o$ such that $Prj_{\Sigma_o}(s) = Prj_{\Sigma_o}(s')$, $T^{md}(q_0^{md}, Prj_{\Sigma_o}(s)) = q^{md}$, $T(q_0, s) = T(q_0, s') = q^o$, $l = P_{\Sigma_f}(s)$, $l' = P_{\Sigma_f}(s')$, and $l \prec\!\succ l'$.*

Definition 7. *A set of δ-incomparable states $q_1^{md} \cdots q_n^{md} \in Q^{md}$ is said to form a δ-indeterminate cycle if $T^{md}(q_i^{md}, \sigma_i) = q^{md}_{(i+1) \bmod n}$, where $\sigma_i \in \Sigma_o$, $i \in [1 \cdots n]$.*

According to Definition 7, we have the following lemma.

Lemma 2. *Let q_1^{md}, q_2^{md}, ..., $q_n^{md} \in Q^{md}$ be a set of δ-incomparable states forming a δ-indeterminate cycle, where*
$q_i^{md} = \{(q^o_{i_1}, l_{i_1}), (q^o_{i_2}, l_{i_2}), \ldots, (q^o_{i_{len\_i}}, l_{i_{len\_i}})\}$,
$q_j^{md} = \{(q^o_{j_1}, l_{j_1}), (q^o_{j_2}, l_{j_2}), \ldots, (q^o_{j_{len\_j}}, l_{j_{len\_j}})\}$,
with $1 \leq i, j \leq n$, and len_i, len_j denoting the number of pairs in q_i^{md} and q_j^{md}, respectively. Then we have:

$$\{l_{i_1}, l_{i_2}, \ldots, l_{i_{len\_i}}\} = \{l_{j_1}, l_{j_2}, \ldots, l_{j_{len\_j}}\}.$$

In other words, any two states in a δ-incomparable cycle share the same set of different fault labels.

Based on above definitions and lemmas, the necessary and sufficient conditions for strong minimal diagnosability of a DES G based on its minimal diagnoser G^{md} are provided below.

Proposition 1. *A language L generated by an FSM G is* strongly minimally diagnosable *iff its minimal diagnoser G^{md} satisfies the following conditions:*

(C_1) *There is no δ-indeterminate cycle in G^{md};*

(C_2) *No state $q^{md} \in Q^{md}$ is ambiguous;*

(C_3) *For each δ-incomparable state $q^{md} \in Q^{md}$ and for each pair $(q^o, l) \in q^{md}$, there exist a state $q^{md'} \in Q^{md}$ and a string $s_o \in \Sigma_o^*$ such that $T^{md}(q^{md}, s_o) = q^{md'}$, and for each pair $(q^{o\prime}, l') \in q^{md'}$, we have $l' = l$, that is, $q^{md'}$ (after q^{md}) is a δ-certain state with the* unique *minimal fault label l.*

Example 5. Consider the four DESs modeled by G_1, G_2, G_3, and G_4 in Fig. 2, respectively, where f_1, f_2, and f_3 are all failure events, and the other events are all observable. Their minimal diagnosers G_1^{md}, G_2^{md}, G_3^{md}, and G_4^{md} are also outlined in Fig. 2. Among the four minimal diagnosers, only G_1 is strongly minimally diagnosable. G_2 is not strongly minimally diagnosable as it does not fulfill Condition (C_1): there is a δ-indeterminate cycle including state $\{(4, \{f_1\}), (5, \{f_2\})\}$ and cyclic transition event o_4 in G_2^{md}. G_3 is not strongly minimally diagnosable as it does not fulfill Condition (C_2): states $\{(4, \{f_1\}), (4, \{f_2\})\}$ and $\{(6, \{f_1, f_3\}), (6, \{f_2, f_3\})\}$ in G_3^{md} are both ambiguous. Also G_4 is not strongly minimally diagnosable as it does not fulfill Condition ($C3$): there is a δ-incomparable state $q^{md} = \{(4, \{f_1\}), (5, \{f_2\})\}$ in G_4^{md}, but there are no states like $\{(4', \{f_1\})\}$ or $\{(5', \{f_2\})\}$ after q^{md} in G_4^{md}. $\diamond$

6 Weak Minimal Diagnosability of DESs

Definition 5 requires that any minimal diagnosis δ must be the *unique* minimal diagnosis after a definite delay, before a new fault (not in δ) occurs. The logical condition is very strong. Thus, we provide the notion of weak minimal diagnosability.

Definition 8. *A prefix-closed and live language L is* weakly *minimally diagnosable if the following condition holds:*

$$\forall \delta (\delta \in L_f)\ \forall s (s \in s_\delta)\ \exists n (n \in \mathbb{N})$$
$$\forall t (t \in L/s)\ (\|t\| \geq n \Rightarrow D_m),$$

where the minimal diagnosability condition D_m is defined as follows:

$$(\delta \models_{min} Prj_{\Sigma_o}(st)) \Rightarrow$$
$$(\omega \in Prj^{-1}_{\Sigma_o}(Prj_{\Sigma_o}(st)) \Rightarrow \delta \preceq P_{\Sigma_f}(\omega)).$$

In other words, let s be a trace generated by G ending with a failure event set δ. For any continuation t of s there exists a natural number n such that, when the length of t is not less than n, if δ is still the minimal diagnosis of st, then δ will be the *unique* minimal diagnosis for any trace with the same observation sequence in st.

If a DES is weakly minimally diagnosable, it is only required that, when the length of a trace is very large (the length of continuation t is greater than n), if the fault set of the trace is minimal, then it will become the *unique* minimal diagnosis definitely. Thus, the logical condition is weaker than that provided in Definition 5 and Definition 1. The following proposition, relating the notion of classical diagnosability and the two notions of minimal diagnosability, holds.

Proposition 2. *Let G be a DES. If G is strongly minimally diagnosable then G is weakly minimally diagnosable. If G is diagnosable then G is weakly minimally diagnosable.*

The contrary of Proposition 2 does not hold, as shown in the following example.

Example 6. The system G_4 in Fig. $2(d)$ is weakly minimally diagnosable, yet not strongly diagnosable, according to their definitions. The system G in Fig. $1(a)$ is weakly minimally diagnosable, yet not diagnosable. $\diamond$

In the following we give the necessary and sufficient conditions for weakly minimal diagnosability of a DES.

Proposition 3. *A language L generated by an FSM G is weakly minimally diagnosable iff its minimal diagnoser G^{md} does not include any δ-indeterminate cycle.*

Example 7. Consider the four DESs and relevant minimal diagnosers in Fig. 2. Based on the minimal diagnosers we can see that G_1 and G_4 are weakly minimally diagnosable. G_2 is not weakly minimally diagnosable, because there is a δ-indeterminate cycle including state $\{(4, \{f_1\}), (5, \{f_2\})\}$ and cyclic transition event o_4 in G_2^{md}. G_3 is not minimally diagnosable, because there is also a δ-indeterminate cycle including state $\{(6, \{f_1, f_3\}), (6, \{f_2, f_3\})\}$ and cyclic transition event o_2 in G_3^{md}. $\diamond$

7 Related Work

Since the seminal work on model-based diagnosis of DESs [Sampath *et al.*, 1995], there have been a lot of *diagnoser*-based approaches for deriving diagnoses and for testing diagnosability, including decentralized/distributed diagnosers and diagnosability [Pencolé and Cordier, 2005; Pencolé, 2004; Ye and Dague, 2010; Moreira *et al.*, 2011] and fuzzy/stochastic diagnosers and diagnosability [Thorsley and Teneketzis, 2005; Liu and Qiu, 2009].

Some other approaches are not based on constructing a diagnoser explicitly, including diagnosis of active systems [Lamperti and Zanella, 2003] and the distributed framework for diagnosis of DESs [Su and Wonham, 2005].

To our knowledge, nearly all of them consider *all* possible diagnoses. Therefore, a (possibly large) number of non-minimal diagnoses are generated, which are less probable or even almost impossible, but at cost of considerable computation time and memory allocation. Instead, in our approach only minimal diagnoses are considered, which is consistent with previous approaches to minimal diagnosis of static systems [Reiter, 1987; de Kleer and Williams, 1987].

Recently, model-based diagnosis with different notions of preference was proposed in [Grastien *et al.*, 2011; 2012]. Also, they gave some approaches to generate all diagnoses online, mainly based on the generate-and-test strategy. By contrast, our minimal diagnosers of DESs can

be built off-line, with all diagnoses being stored in them, a valuable property for online diagnosis.

There are several works aimed at finding minimal diagnosis based on either AI planning (e.g. [McIlraith, 1994; Sohrabi *et al.*, 2010]) or SAT approaches (e.g. [Haslum and Grastien, 2011; Grastien *et al.*, 2007]), but not on *diagnosers*. However, a diagnosis problem has to be first transformed into the corresponding planning or SAT representation (most manually, not automatically at least now by our knowledge). Moreover, they generally have the bottleneck for very quickly solving SAT or planning problems for online diagnosis.

In addition, some other definitions of diagnosability are also proposed from different viewpoints, including distributed diagnosability [Pencolé, 2004], decentralized diagnosability [Debouk *et al.*, 2000; Wang *et al.*, 2007], modular diagnosability [Contant *et al.*, 2006], and weak diagnosability [Rozé and Cordier, 2002]. However, all of them are based on classical (non-minimal) diagnosis.

8 Conclusion

In order to compactly represent the more probable diagnostic results of a DES, a kind of minimal diagnosis is proposed, where only the fault sets with minimal cardinality are considered. Furthermore, a type of minimal diagnoser is presented for online minimal diagnosis. Finally, two minimal notions of diagnosability are introduced for determining whether a DES is minimally diagnosable.

Remark *All lemmas and propositions can be formally proven; in this paper proofs are omitted for space reasons.*

References

[Cassandras and Lafortune, 2008] G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2008.

[Contant *et al.*, 2006] O. Contant, S. Lafortune, and D. Teneketzis. Diagnosability of discrete event systems with modular structure. *Discrete Event Dynamic Systems*, 16:9–37, 2006.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[Debouk *et al.*, 2000] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosisof discrete event systems. *Discrete Event Dynamic Systems*, 10:33–86, 2000.

[Grastien *et al.*, 2007] Al. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, pages 305–310, Vancouver, Canada, 2007.

[Grastien *et al.*, 2011] Al. Grastien, P. Haslum, and S. Thiébaux. Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 60–67, Murnau, Germany, 2011.

[Grastien *et al.*, 2012] Al. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: theory and practice. In *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR-12)*, pages 489–499, Rome, Italy, 2012.

[Haslum and Grastien, 2011] P. Haslum and Al. Grastien. Diagnosis as planning: two case studies. In *Proceedings of the 5th Scheduling and Planning Applications Workshop (SPARK-11)*, pages 37–44, Freiburg, Germany, 2011.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of Active Systems: Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[Liu and Qiu, 2009] F. Liu and D. Qiu. Diagnosability of fuzzy discrete-event systems: A fuzzy approach. *IEEE Transactions on Fuzzy Systems*, 17:372–384, 2009.

[McIlraith, 1994] S. McIlraith. Generating tests using abduction. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 449–460, Bonn, Germany, 1994.

[Moreira *et al.*, 2011] M. V. Moreira, T. C. Jesus, and J. C. Basilio. Polynomial time verification of decentralized diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 56:1679–1684, 2011.

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

[Pencolé, 2004] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI-04)*, pages 22–27, Valencia, Spain, 2004.

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

[Rozé and Cordier, 2002] L. Rozé and M.-O. Cordier. Diagnosing discrete-event systems: Extending the "diagnoser approach" to deal with telecommunication networks. *Discrete Event Dynamic Systems*, 12:43–81, 2002.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40:1555–1575, 1995.

[Sohrabi *et al.*, 2010] S. Sohrabi, J. Baier, and S. McIlraith. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR-10)*, pages 26–36, Toronto, Canada, 2010.

[Su and Wonham, 2005] R. Su and W. M. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control*, 50:1923–1935, 2005.

[Thorsley and Teneketzis, 2005] D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50:476–492, 2005.

[Wang *et al.*, 2007] Y. Wang, T.-S. Yoo, and S. Lafortune. Diagnosis of discrete event systems using decentralized architectures. *Discrete Event Dynamic Systems*, 17:233–263, 2007.

[Ye and Dague, 2010] L. Ye and P. Dague. Diagnosability analysis of discrete event systems with autonomous components. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, pages 105–110, Lisbon, Portugal, 2010.

Exploiting Passing Tests for Value-Level Debugging of Verilog Designs

Bernhard Peischl[1,3] and **Franz Wotawa**[2,3]
[1]Softnet Austria
email: bernhard.peischl@soft-net.at
[2]Institut für Softwaretechnologie, Technische Universität Graz
email: wotawa@ist.tuGraz.at

Abstract

In this article we report on novel insights in model-based software debugging of hardware description languages (HDLs). Our debugging model allows one for exploiting failing and passing test cases by incorporating Ackermann constraints. This article reports on an empirical evaluation of the introduced models. The evaluation of our approach on the well-known ISCAS 89 benchmarks concerning single and dual-fault diagnoses clearly indicates that incorporating passing test cases into fault localization considerably improves the accuracy of the obtained diagnosis candidates.

1 Introduction

[1]This article reports on the most recent results in software debugging of Verilog designs. It is a major extension to previous research work (Peischl and Wotawa 2006) that primarily reports on fault localization in VHDL (Very High Speed Integrated Hardware Description Language) (Navabi 1993, IEEE 1988). Verilog (IEEE 1995), has a formal semantics and thus is amendable to research in verification and debugging, e.g., its synthesis semantics is formally specified in Gordon (Gordon 1995, Gordon 2002).

Most of the research in verification deals with the detection of faults and does not address the fact that debugging involves locating and correcting the fault. In detecting faults (software/hardware testing), we make use of numerous test cases for more than two decades. In the recent past, numerous test cases have been employed for localizing faults e.g. in terms of employing spectrum-based diagnosis (Abreu et al. 2007, Baudry et al. 2006, Hao et al. 2005, Hao et al. 2009, Liblit et al. 2005, Yu et al. 2008).

Spectrum-based techniques, however, allow one for logical reasoning at the level of dependencies and do not consider the semantics of the language in terms of value-level models. Consequently there is a lack of research dealing with multiple test cases in conjunction with value-level models (Peischl and Wotawa 2006) taking into account language semantics. This is noteworthy as we do have well-founded techniques that allow for considering whole test suites and – as shown in this article – there is solid empirical evidence that taking into account test suites improves the fault localization capabilities considerably.

Over the last 25 years, the Artificial Intelligence community has developed a framework for system diagnosis called model-based diagnosis (MBD). This framework is extremely general and covers a broad range of capabilities including the isolation of faulty components and the handling of multiple fault locations (Reiter 1987, de Kleer et al. 1990). Harnessing these techniques in software engineering tools considerably may help to master the development of complex circuits and software-enabled systems.

Since its well-founded theory we rely on MBD, and employ the ISCAS 89 benchmark suite (Brglez 1989) to demonstrate the practical applicability of our novel models. Relying on an exhaustive evaluation our insights clearly indicate, that the incorporation of test suites (rather than only single test cases as for example in (Peischl and Wotawa 2006, Wotawa 2002)) considerably contributes to accurately locate the root cause for detected misbehavior. In this paper we show how to exploit passing test cases. As these test cases do not contribute to further conflicts, these tests have not been used in previous work (Peischl and Wotawa 2006) in the process of computing diagnoses from conflicts. According to our empirical evaluation using the ISCAS 89 benchmarks, with a couple of failing test cases (up to 5), we can exclude almost 94 percent of the statements and expressions of being faulty. By leveraging passing test cases, we further can rule out around half of the remaining 6% of the potentially erroneous code. In this article we show how to incorporate passing test cases and report on our most recent empirical evaluation on the ISCAS 89 benchmarks.

2 Simulation, Test and Debugging

In designing circuits, a designer starts with an initial specification that primarily captures the functional requirements for the circuit being designed. Usually this is followed by a detailed design on the register transfer level (RTL). Both designs are executable and thus are amendable to automated verification. In general, the RTL design is verified very thoroughly in terms of testing and various other analysis techniques, e.g. hazard analysis. Since there is a fixed win-

[3]We listed the authors in alphabetical order.

dow for start of production, these verification steps typically are conducted under time pressure and thus the time for debugging – detecting, localizing, and repairing the misbehavior – is a critical process measure.
Typically, the design process iterates through several steps: Design and programming is followed by a simulation of the circuit. The outcome of the simulation is compared to the specification, that is, it is checked whether the waveform traces on a higher abstraction level (the specification) deviate from the waveforms obtained from the test run on the RTL level. Previous research work, carried out in the VHDL domain, gives an intuitive understanding on how to leverage MBD for fault localization in HDL designs (see a brief video at www.ist.tugraz.at/staff/peischl/HDLDebugging.wmv).
According to a study conducted at IBM Haifa, 50 to 80 percent of the overall development is attributed to verification activities, and localization and correction amounts to 35 percent of the design cycle (Auerbach et al. 2005). Thus, particularly under local or temporal separation of the design and the test team, the automation of fault localization (and correction) is a sustainable topic for ongoing and future R&D work as it contributes to make the development process more efficient.

3 Debugging Sequential Verilog Designs

The semantics of Verilog has been analyzed rigorously, and thus provides the necessary theoretical underpinning in language semantics and circuit synthesis. Gordon (Gordon 2002) provides a formal description of various semantic interpretations of Verilog like event-semantics and trace-semantics. In event-semantics (which is the semantics employed for fine-grained simulations) the change of a variable necessitates the recalculation of depending procedures.
In contrast to that, the trace semantics of Verilog computes solely the quiescent states at the end of a simulation cycle. For computing these quiescent values, each procedure is evaluated only once per cycle (Gordon 1995). Procedures are evaluated in an order such that a procedure is not evaluated until all its driving procedures have been evaluated. In other words, the outputs of a procedure are computed only when all its inputs are known (or already computed). So we build up our representation of the design by starting with processes solely dependent on known inputs and variables (e.g., the primary inputs, including clock). Afterwards, the outputs of these processes are attached to the list of already known inputs and variables. This process continues until all the procedures in the design are levelized (Peischl and Wotawa 2006). In this way we build up a chain of procedures and their inputs and outputs, thus allowing for an evaluation of all the variables used in the design at the end of the simulation cycle.
Synchronous sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a clock signal. In other words synchronous sequential circuits consist of multiple cycles. In electrical engineering, sequential circuits are often viewed as a sequence of connected combinational circuits. This can be done by selecting some connections and splitting them in two separated connections. One is the input and one the output. The output of a stage of a specific cycle is connected to the corresponding input of the next cycle.
We have adopted the same idea for providing an appropriate debugging model for sequential designs. Our representation can be broken into two phases, one in which latches change state, and one in which all the combinational blocks are evaluated. We effectively break the design at latches by treating the outputs of the latches as they were inputs and inputs of the latches as they were outputs.
In our representation, we first identify variables that we have to synthesize into latches. By splitting these variables and treating them as additional inputs and outputs, we ensure that our representation remains acyclic. Then we levelize the graph according to the levelization strategy discussed above. Thus we receive a sequence of procedures depicting the data flow from the given primary inputs to the primary outputs. Our next step is to unroll the sequential circuits to incorporate multiple cycles (input sequence length). We assume that we know the number of unrollings to be performed in advance. After the levelization of all the procedures, we create the component-connection model. This component-connection model (Reiter 1987, de Kleer 1990) represents our model at level 1 (cycle no. 1). For every component *C*, we attach a timestamp *i* during the creation of the model to ensure a unique identification. Thus C_i represents the instance of component *C* at cycle *i*. Thus we make *n* copies of every component involved, where *n* is the total number of cycles or unrollings. So we create *n* number of instances for each component.

Diagnosis problem: A diagnosis problem considering circuit unrolling over *n* cycles is a triple *(SD, COMP,OBS)* where

- $SD = \bigcup_{i=1..n} SD_i$ where SD_i is the system description for cycle i
- $COMP = \bigcup_{i=1..n} C_i$ where C_i are the components in cycle i, and
- $OBS = \bigcup_{i=1..n} OBS_i$ and OBS_i denote the observations in cycle *i*.

The above given definition captures a diagnosis model for a single test case (of length *n*). Given this definition the diagnosis problem considering a test suite is given as follows:

Diagnosis problem, test suite: Given a test suite comprising the test cases TC_1, TC_2, …, TC_k. Let the system description SD_j be the system description considering test case TC_j and let C_i^j be the instance of component *C* at cycle *i* in test case number *j*. Correspondingly OBS_i^j denote the observations in cycle *i* of test case TC_j. The diagnosis problem $(SD^*, COMP^*, OBS^*)$ considering this test suite is given as follows:

- $SD^* = \bigcup_{j=1..k} SD_j \cup \{\neg AB(C_0^j) \rightarrow \neg AB(C_1^j) \wedge \neg AB(C_2^j) \wedge \ldots \quad \ldots \wedge \neg AB(C_n^j)\}$
- $COMP^* = \bigcup_{j=1..k, i=0..n} C_i^j$
- $OBS^* = \bigcup_{j=1..k, i=1..n} OBS_i^j$

As passing testcases do not cause a logical contradiction, we do not obtain conflicts from passing testcases considering the diagnosis model for a test suite *(SD*, COMP*, OBS*).*

4 Exploiting Passing Testcases

To illustrate the potential of using passing test cases to locate the root cause for detected misbehavior we continue with a simple example.

assumption	in1	in2	out	inter	verdict
AB(not), ¬AB(xor)	1	0	1	0	fail
AB(not), ¬AB(xor)	0	0	1	1	pass

Figure 1: Passing and failing testcases and part of a circuit.

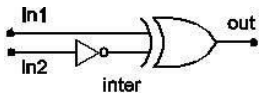


Figure 1 illustrates a part of a circuit an exclusive or and a not gate together with a passing and failing test case. We further assume that the circuit is faulty, that is, our test suite has identified misbehavior and we obtain both components (the exclusive or and the not gate) as possible diagnosis candidates.

Suppose we have the test cases given in Figure 1. Considering the first (failing) test case in the first line, and assuming the not gate to be abnormal but the exclusive or gate to be correct, we can deduce that signal inter becomes 0. However, under the same assumption, the passing test case in line 2, forces the value of inter to become 1. We immediately see that the not gate is required to map the signal inter to 0 and to 1 for the same input value $in_2=0$. Obviously, no deterministic component can fulfill this requirement. Thus the not gate can no longer be considered as a valid diagnosis candidate. To our best knowledge, the authors of (Raiman et al. 1991) were the first who used this idea for discriminating diagnosis candidates. Unfortunately, the article gives no further insights whether the technique can be employed in practice as the authors do not provide an empirical evaluation to evaluate scalability and the improvement with respect to accuracy.

In the following we propose an extension to that which – under absence of structural faults – allows one for taking advantage of passing test cases. As passing test cases do not yield to additional conflicts, we capture the specific information about diagnoses in terms of Ackermann constraints (Ackermann 1954). By adding these consistency constraints we incorporate the fact that the same combination of input values applied to a deterministic component *C* produces the same output for every instance of *C*. This allows for exploiting the many test cases that typically do not reveal a fault. The system description with Ackermann constraints SD_A is given as follows:

System description with Ackermann constraints: Let TC_p be a set of passing test cases from a test suite *TC*, let $in(C_i) = \{ i_{Ci}^1, \ldots, i_{Ci}^m \}$ denote the inputs of component C_i, let $out(C_i) = \{ o_{Ci}^1, \ldots, o_{Ci}^n \}$ denote the outputs and let SD^* denote the system description of a diagnosis problem considering a test suite. The system description with Ackermann constraints SD_A is given by,

$$SD_A = SD^* \cup CON_A,$$

$$CON_A = \neg AB(C_i) \wedge \forall_{l=1}^m i_{ci}^l = i_{cj}^l \rightarrow \forall_{p=1}^n o_{ci}^p = o_{cj}^p$$

where, i≠j and i,j denote indices of the passing test cases.

As we will show in the next section, Ackermann constraints increase the complexity of the model considerably. Therefore, we used a post processing technique proposed by the authors of (Wotawa 2002). As shown at the end of this section filtering allows one for iteratively applying the Ackermann constraints to the obtained diagnoses. Instead of compiling the constraints into the debugging model, we apply the constraints in terms of a dedicated post-processing phase.

Filtering refers to discarding certain diagnoses by taking advantage of further test cases TC_i. A diagnosis Δ 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 $C \in \Delta$ that allows for repairing the program for the given test case. The function *replace(C)* allows for producing the correct output values for the considered test case. However, considering a test suite such a replacement does not exist for all test cases in the test suite TC necessarily.

Since all components $COMP \setminus \Delta$ are assumed to behave correctly, we can compute the input values *in(C)* and *out(C)* for every component *C* from *Δ* (employing forward propagation). According to this computed input/output relation, the component *C* may be required to map the same input- to different output values. This corresponds to an inconsistency and the specific diagnoses *AB(C)* is not repairable wrt. the specific test case. As there is no function *replace(C)* as stated previously, the component *C* can be removed from the set of diagnosis candidates. In this vein, we evaluate the Ackermann constraints in an iterative way by checking for different input values for a certain output value.

Algorithm 1 (Filtering): Let Δ denote a set of diagnosis candidates and let TS be a test suite.

1. *For all* $D \in \Delta$ *do*
2. *For all test cases* $TC_i \in TC$ *do*
   a. *Let* i_{Di} *denote the input values and let* o_{Dj} *denote the output values of component D by assuming* $AB(D) \wedge \{\neg AB(C) \mid C \in COMP \setminus D\}$
   b. *If there exits i,j, i≠j, such that* $i_{Di} = i_{Dj} \wedge o_{Di} \neq o_{Dj}$ *then remove D from Δ*
3. *return Δ*

Claim: Algorithm 1 applies the Ackermann constraints CON_A to a set of single-diagnosis candidates.

After applying Algorithm 1 to the set of single-fault diagnosis candidates, there is no component D at which we obtain different input values for a certain output value. Thus we conclude, that

$$\neg\exists i, j, i \neq j \bullet (\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \wedge (\forall_{p=1}^{n} o_{Di}^{p} \neq o_{Dj}^{p})$$

$$\forall i, j, i \neq j \bullet \neg(\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \vee (\forall_{p=1}^{n} o_{Di}^{p} = o_{Dj}^{p})$$

$$\forall i, j, i \neq j \bullet (\forall_{l=1}^{m} i_{Di}^{l} = i_{Dj}^{l}) \rightarrow (\forall_{p=1}^{n} o_{Di}^{p} = o_{Dj}^{p})$$

Algorithm 1 thus imposes the Ackermann constraints on the set of single-fault diagnosis candidates. For our evaluation of the approach we therefore took advantage of the filtering algorithm presented previously.

5 Practical Experiences and Evaluation

With a series of our most recent experiments we pursue the goal to evaluate the discriminating capabilities of several test cases on sequential circuits, the response time (and thus the computational complexity on a technical level) and the effect of the filtering technique.

We conducted our experiments on a Dell Power Edge 1950 II - 2x Quad Core with 2.0 GHz and 10GB of RAM. For computing diagnoses we relied on the extension of Reiter's algorithm described in (Peischl et al. 2013). Note that, for the efficient computation of diagnoses, we convert the rules capturing the language semantics (discussed in Peischl et al. 2012) into a specific Horn-like encoding (Peischl and Wotawa 2003). As the computation of conflict sets is a time critical issue, the (minimal) conflict sets are computed according to the procedure explained in (Peischl and Wotawa 2003). The diagnosis engine and the proposed extension are implemented in the Java programming language.

Our debugging tool parses the Verilog code, builds up the model as described in this article and converts a test suite to the logical representation (Peischl et al. 2012). Afterwards the tool computes diagnosis candidates in increasing order of cardinality and visualizes the results by highlighting the corresponding statements, expressions or operators.

5.1 Time Complexity of Computing Diagnosis

For our empirical evaluation we use a Horn-like encoding of the rules presented herein. By relying on this encoding we make use of an efficient procedure to compute all minimal conflicts (Peischl and Wotawa 2003). From the obtained conflicts we retrieve diagnoses by computing the minimal hitting sets in increasing order, where for practical purposes, primarily single- and double-fault diagnoses are of interest. In general searching for all diagnoses has a worst time complexity of the order $O(|MODES|*|COMP|^{s})$, where |MODES| is the number of fault modes, |COMP| is the number of components and s is the maximal size of the diagnoses (Wotawa 1996). Since we use two fault modes (AB(C) and $\neg$AB(C)) and search for single and double fault diagnoses, our worst time complexity is of the order $O(|COMP|^{2})$. Note that we consider the components in every cycle as independent and thus the number of components increases with the length of the test case. However, the average running time complexity is much better because diagnoses with smaller size (particularly single-fault diagnoses) are more likely than diagnoses with bigger size. For example, finding all single diagnoses is of order O(|COMP|) assuming the decision procedure can be executed in unit time.

5.2 Test Suite Generation

We obtained the test suite by injecting a single-fault (respectively a dual-fault for the second series of experiments) into the RTL design. We introduced a random fault and used circuit equivalence checking to generate test cases revealing this fault. We stopped the generation process as soon as we obtained five test cases revealing the introduced fault. In some (rare) cases, for example for the circuit s444, we were not able to find five test cases and stopped this process earlier (see Figure 2). The faults are introduced in a random way by picking a statement from every circuit and replacing this statement by another statement. That is, for every circuit, we replaced an arbitrary statement with a structurally equivalent statement (same no. of input parameters).

For example, in a specific circuit we randomly selected a NOR statement and replaced it by an AND statement. Further we implicitly removed/added negations as we substituted a logical statement by the negated counterpart (e.g., NAND by AND vice versa). These error types are not necessarily complete wrt. functional errors, but as they are believed to be common in the design process, we capture the most common scenarios (Nayak and Walker 1999): (1) Mistakenly replacing one gate by another gate with the same number of input and (2) incorrectly adding or removing a gate.

All empirical evaluations are conducted on the Verilog RTL version of the ISCAS 89 benchmark suite (Brglez et al., 1989). Further, the gate-level representations of the ISCAS 89 benchmarks have been used to obtain the correct waveform traces since our simulator allowed only for simulation of gate-level circuits. A detailed analysis including the results for the specific circuits can be found in (Peischl et al. 2012). In the following we summarize the major results.

5.3 Empirical Evaluation and Discussion

In our experimental setting we assumed that an engineer only knows the correct values of the primary inputs for every simulation cycle and the outputs at the end of the final simulation cycle. That is, the traced variables correspond to the primary inputs v_{in} for every instant of time (v_{in}, val_{in}, t), $t=1..n$, together with the primary outputs (v_{out}, val_{out}, n) at time n and thus the observations are given in terms of the primary input variables for every cycle and the primary output variables at the end of the simulation cycle (i.e. at time point n, where n is the length of the test case). To evaluate the impact of the temporal unfolding of the circuit we conducted experiments with four and eight simulation cycles relying on the well-known ISCAS 89 benchmark suite.

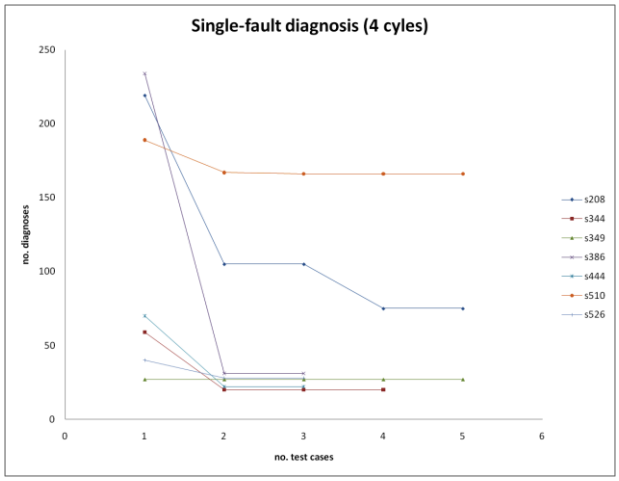


Figure 2: No. of obtained single-fault diagnoses for the ISCAS 89 benchmark (4 cycles).

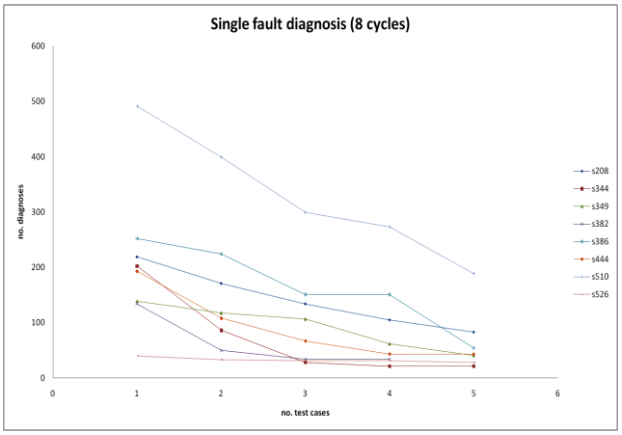


Figure 3: No. of obtained single-fault diagnoses (ISCAS 89, 8 cycles).

First, the figures underpin the findings discussed in previous research papers (Friedrich et al., 1999), as the number of single diagnoses being obtained depends from both, the concrete test case and the structural complexity of the program. Second, as Figure 2 and 3 illustrate – even with only a couple of test cases (in our case up to 5) – the number of obtained diagnoses can be reduced significantly when compared to the experiment with solely a single test case. Remarkably, the random fault introduced in circuit s510 yields to a significant number of diagnoses and thus higher response times when compared to the remaining circuits. It appears, that (1) the structural complexity, (2) the specific error being introduced and the (3) specific test cases identifying the introduced faults result in a (at least in relation to the other circuits) computationally expensive problem.

On average we obtained 74(123) single-fault diagnoses and 44(70) faulty lines in the source code when unfolding the circuit for 4(8) instances of time. Remarkably, a designer can exclude over 90 percent of the source code from being faulty (93,6 percent for 4 cycles and 92,5 percent for 8 cycles of unfolding).

Figure 4 outlines further empirical results we obtained from the ISCAS 89 benchmark suite considering dual-fault diagnoses as well. When considering dual-fault diagnoses, the no. of diagnosis candidates does not necessarily decrease monotonically with the increasing set of test cases.

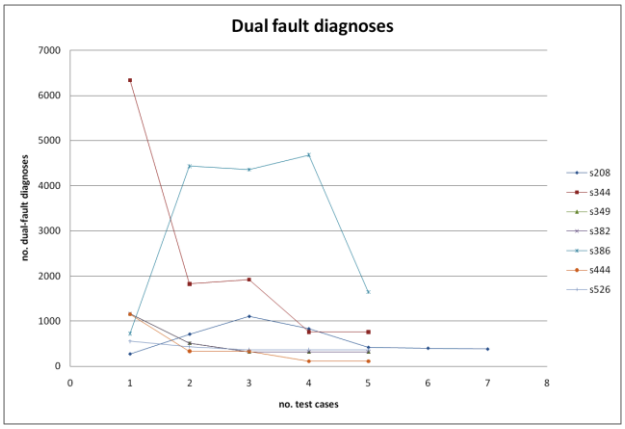


Figure 4: No. of obtained dual-fault diagnosis (ISCAS 89, 4 cycles).

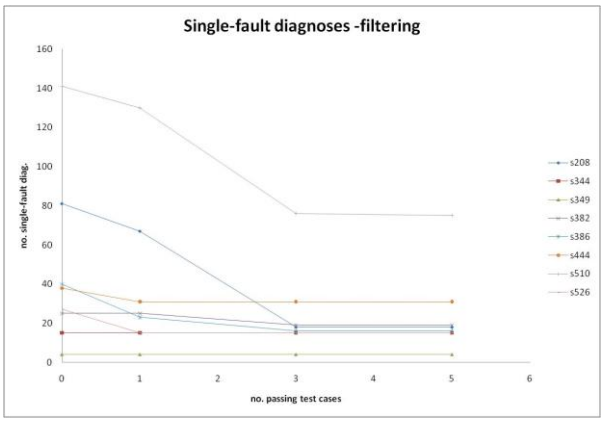


Figure 5: No. single-fault diagnoses when using the filtering algorithm (4 cycles).

However, our experiments revealed that for most of the circuits, the obtained number of fault candidates decreases monotonically with an increase in the size of the test suite. Together with the results for single-fault diagnoses, this gives empirical evidence that the additional cost in the running time, pays off in terms of a higher accuracy in the obtained diagnosis candidates. In (Peischl et al. 2013) we present novel algorithms and an analysis on scalability and the corresponding running times.

Figure 5 summarizes the results on a further series of experiments incorporating the filtering algorithm. To our best knowledge, the filtering approach has never been subject to an empirical evaluation. When compared to Figure 2, one can see that exploitation of passing test cases further contributes to accurately isolate the real cause of misbehavior.

6. Conclusion

In this article we briefly discuss the simulation-driven design process with hardware description languages (HDLs) and point out the importance of fault localization techniques. Afterwards we introduce a model extension that allows one for exploiting failing and passing test cases. Failing test cases result in conflicts and thus contribute to accurately locate the fault. To exploit the numerous passing test cases, we introduce Ackermann constraints and establish a relationship to the filtering technique proposed earlier. Our empirical evaluation on the ISCAS 89 bench-

mark suite demonstrates that the proposed technique is practically feasible and considerably contributes to locate the real cause of misbehavior. According to our experiments using the ISCAS 89 benchmarks, on average we can exclude almost 94 per cent of the statements and expressions from being faulty making use of up to 5 failing test cases per circuit. By leveraging passing test cases, we further are able to rule out around half of the remaining 6 per cent of the potentially erroneous code. These results further motivate research on value-level models for debugging HDL designs. Future research should apply the proposed techniques to even bigger circuits (e.g. using more recent benchmarks etc.) and investigate the relationship between filtering and Ackermann constraints under presence of multiple-fault diagnoses.

References

[Auerbach, 2005] Gadiel Auerbach, Mark Moulinn, Barbara Jobstmann, Roderick Bloem, Alessandro Cimatti, Marco Roveri, PROSYD: Property-Based System Design, Deliverable 2.1/1, May 2005, PROSYD Technical Report, FP6-IST-507219.

[Abreu, 2007] Abreu, R.; Zoeteweij, P.; van Gemund, A.J.C., On the Accuracy of Spectrum-based Fault Localization, Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, vol., no., pp.89-98, 10-14 Sept. 2007.

[Ackermann, 1954] W. Ackermann, Solvable Cases of Decision Problems, North Holland, 1954.

[Baudry, 2006] Benoit Baudry, Franck Fleurey, and Yves Le Traon. 2006. Improving test suites for efficient fault localization. In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 82-91.

[Brglez, 1989] F. Brglez, D. Bryan, and K. Kozminski, Combinational Profiles of Sequential Benchmark Circuits, IEEE International Symposium on Circuits and Systems, 1989.

[de Kleer, 1990] J. de Kleer, A.K. Mackworth, and R. Reiter, Characterizing diagnoses, in Proceedings of the National Conference on Artificial, Intelligence (AAAI), Boston, Aug. 1990, pp. 324–330.

[Friedrich et al., 1999] G. Friedrich, M. Stumptner and F. Wotawa, Model-based diagnosis of hardware designs, Artif. Intell. 111(1-2) (1999) 3-39.

[Gordon, 1995] Michael J. C. Gordon, The semantic challenge of Verilog HDL. In Proceedings of tenth Annual IEEE Symposium on Logic in Computer Science, pages 136–145, 1995.

[Gordon, 2002] Michael J. C. Gordon, Relating event and trace semantics of hardware description languages, The Computer Journal, 45(1):27–36, 2002.

[Hao, 2009] Dan Hao, Lu Zhang, Tao Xie, Hong Mei, and Jia-Su Sun. 2009. Interactive fault localization using test information. J. Comput. Sci. Technol. 24, 5, September 2009, 962-974.

[IEEE, 1995] IEEE Standard Verilog Language Reference Manual LRM Std 11364-1995, Institute of Electrical and Electronics Engineers, Inc. IEEE, 1995.

[Liblit, 2005] Ben Liblit, Mayur Naik, Alice X. Zheng, Alex Aiken, and Michael I. Jordan. 2005. Scalable statistical bug isolation. In Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI '05). ACM, New York, NY, USA, 15-26.

[Navabi, 1993] Z. Navabi, VHDL Analysis and Modeling of Digital Systems, McGraw-Hill, New York, 1993.

[Nayak, 1999] D. Nayak, D. M. H. Walker; Simulation-based design error diagnosis and correction in combinational digital circuits, VLSI Test Symposium, Proceedings of the 17th IEEE Test Symposium (VIS 99), vol., no., pp. 70-79, 1999.

[Peischl and Wotawa, 2003] Bernhard Peischl and Franz Wotawa, Computing Diagnosis Efficiently: A Fast Theorem Prover for Propositional Horn Theories. In Proceedings of the 14th International Workshop on Principles of Diagnosis (DX-03), pages 175-180, Washington DC, June 2003.

[Peischl and Wotawa, 2006] Bernhard Peischl and Franz Wotawa, Automated Source-Level Error Localization in Hardware Designs, pp. 8-19, IEEE Design and Test of Computers, January/February, 2006.

[Peischl et al., 2012] B. Peischl, N. Riaz, F. Wotawa, Automated Debugging of Verilog Designs, International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Sept. 2012, World Scientific.

[Peischl et al., 2013] B. Peischl, N. Riaz, F. Wotawa, Advancements in Automated Debugging of Verilog Designs, Submission to the Applied Intelligence Journal in preparation.

[Raiman, 1991] Olivier Raiman, Johan de Kleer, Vijay Saraswat, and Mark Shirley, Characterizing non-intermittent faults, pages 849–854, In Proceedings AAAI, Anaheim, July 1991, Morgan Kaufmann.

[Reiter, 1987] R Reiter, A theory of diagnosis from first principles, 57-95, Artif. Intell. 32, April 1987.

[Wotawa, 1996] Franz Wotawa, Applying Model-Based Diagnosis to Software Debugging of Concurrent and Sequential Imperative Programming Languages, PhD thesis, Technische Universität Wien, 1996.

[Wotawa, 2002] Franz Wotawa, Debugging hardware designs using a value-based Model, pages 71–92, Applied Intelligence, 16(1), 2002.

[Yu, 2008] Yanbing Yu, James A. Jones, and Mary Jean Harrold. 2008. An empirical study of the effects of test-suite reduction on fault localization. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 201-210.

Using Genetic Algorithms to Study the Effects of Topology on Spectrum Based Diagnosis

Cuiting Chen, Hans-Gerhard Gross and Andy Zaidman
Delft University of Technology, the Netherlands
e-mail: {cuiting.chen; h.g.gross; a.e.zaidman}@tudelft.nl

Abstract

Spectrum-based fault localization (SFL) is a statistical fault diagnosis technique that infers diagnoses from runtime observations. It works by monitoring system transactions, and comparing activity information with pass/fail observations. SFL requires the monitors, which recover the activity data, to be organized to produce optimal information for the diagnosis. This organization is termed topology.

Optimality of monitoring topology for diagnosability represents a search or optimization problem amenable to be addressed by meta-heuristic algorithms. In order to study the effects of topology on the production of diagnoses through SFL, we use genetic algorithms (GA) to generate topologies that lead to improved diagnosability. We illustrate how monitoring topologies affect the diagnosability of systems, and how GA can help to study these effects. We derive general characteristics of topologies to facilitate SFL-based diagnoses.

1 Introduction

Spectrum-based fault localization (SFL) is a lightweight statistics-based automatic diagnosis approach that can be applied to identify misbehaving system parts [5]. It works by automatically inferring a diagnosis from symptoms [1]. The diagnosis is a ranking of potentially faulty system components and the symptoms are observations about component involvement in system activation, plus pass/fail information for each activation [8]. The activation of the system is expressed in terms of a binary activity matrix representing for each component whether it has been involved in a transaction. The pass/fail information is expressed in terms of a binary output vector. A diagnosis is determined by calculating the similarity between each component's activation vector and the output vector. A component whose activity vector is more similar to the output vector is more likely faulty than other components, and ranked higher as suspect.

The application of SFL creates a particular challenge, i.e. the placement of the monitors for gathering component involvement information. We refer to this placement as the *monitoring topology* of the diagnosis system. In principle monitors may be placed anywhere in the monitored system. However, the places should be selected carefully to yield the best results in terms of calculating correct diagnoses. Typical places are in or around the system components, or collections of system components, or between them. Finding monitoring topologies that lead to high diagnosability represents a difficult optimization problem amenable to be solved by meta-heuristic algorithms, such as genetic algorithms. This brings us to the formulation of the following research questions:

RQ1: How can genetic algorithms be used to determine better diagnosable topologies?

RQ2: What are characteristics of topologies that are better diagnosable?

One contribution of this paper is the application of GA, including the definition of adequate fitness functions, in order to study the optimality of topologies for better diagnosability. Another contribution is the formulation of general characteristics of topologies that improve SFL-based diagnoses. Optimization of topology is a well-known problem domain to be addressed by genetic algorithms, e.g. [4], however, the use of GA in spectrum-based software fault localization is novel, in particular the formulation of the fitness introduced.

The remainder of this article is organized as follows. Section 2 introduces SFL and how it is affected by topology. Section 3 illustrates how GA can be applied for SFL topology optimization. Section 4 outlines our experiments performed, and Section 5 presents the discussion of their results, and lessons learned. Finally, Section 6 lists the related work, and Section 7 summarizes and concludes the paper and gives an outlook on future work.

2 Background and Scope

Spectrum-based fault localization calculates a diagnosis ranking of potentially faulty components from observing their activity and pass/fail outcome [8]. Activity is expressed in terms of block-hit-spectra [12], producing per transaction a binary coverage spectrum [21][23] and a verdict. Component activity and verdicts are derived through dedicated monitors. This is demonstrated in [5].

Table 1 illustrates SFL with a system made of components $C_0 - C_{10}$. It is activated with 6 transactions $t_1 - t_6$, leading to the corresponding component activations in the activity matrix. Four transactions

Table 1: Illustration of SFL

Cmp	Character counter	t_1	t_2	t_3	t_4	t_5	t_6	SC_o
	def count(string)	[Activity Matrix]						
C_0	let = dig = other = 0	1	1	1	1	1	1	0.82
C_1	string.each_char { |c|	1	1	1	1	1	1	0.82
C_2	if c===/[A-Z]/	1	1	1	1	0	1	0.89
C_3	**let += 2**	1	1	1	1	0	0	1.00
C_4	elsif c===/[a-z]/	1	1	1	1	0	1	0.89
C_5	let += 1	1	1	0	0	0	0	0.71
C_6	elsif c===/[0-9]/	1	1	1	1	0	1	0.89
C_7	dig += 1	0	1	0	1	0	0	0.71
C_8	elsif not c===/[a-zA-Z0-9]/	1	0	1	0	0	1	0.58
C_9	other += 1 }	1	0	1	0	0	1	0.58
C_{10}	return let, dig, other	1	1	1	1	1	1	0.82
	end							
	Output vector (verdicts)	1	1	1	1	0	0	

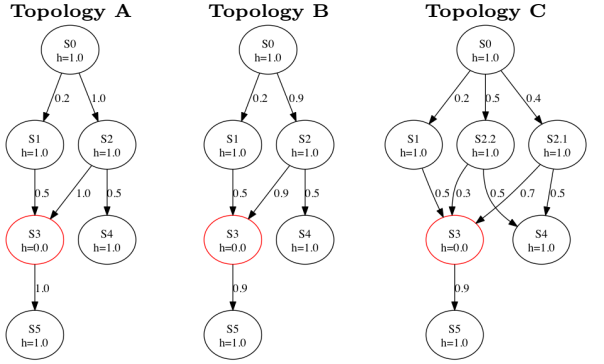


Figure 1: Different diagnosis topologies of an example service-based system

are failing, two are passing, noted in the output vector. The Ochiai similarity coefficient (SC_o) is determined for each component activation vector (a_i) and the output vector (o_i). SC_o is based on three counters $n(1,1)$, $n(1,0)$, $n(0,1)$, representing the respective numbers of occurrences that a_i and o_i form the combinations $(1,1),(1,0),(0,1)$, and it is defined by: $SC_o = n_{11}/\sqrt{(n_{11}+n_{01})\cdot(n_{11}+n_{10})}$. This denotes the likelihood of a component being faulty and determines its position in the ranking. Any SC may be used; however, the Ochiai SC has been found to work best [2]. By applying SC_o in Table 1, C_3 is correctly identified as the faulty component in this example system (faulty component marked bold in Table 1).

Table 2 illustrates how monitoring topology affects SFL. The three topologies shown are comprised of six components, C_1 – C_6. In Topology A and B, every component represents a monitor collecting component activation information. In Topology C, component C_3 is split to represent two monitors, i.e. $C_{3.1}$ and $C_{3.2}$. Component C_4 is faulty with health h=0.0, all other components are healthy (h=1.0). The invocation probabilities between the components are represented by the numbers noted down at the arrows. Topology A shows a particular system characteristic, with components C_1, C_3, C_4, and C_6 being tightly coupled, indicated by the 1.0 invocation probabilities. This represents a specific inhibiting factor for the calculation of the diagnosis as shown by the poor component ranking in the corresponding activity matrix. Components C_1, C_3, C_4 get attributed the same ranking, rendering this diagnosis ambiguous. Topology B represents a relaxation of the tight couplings between components C_1, C_3, C_4, and C_6 to a lower value of 0.9. This leads to a better SC_o calculation and a correct diagnosis of the faulty service for Topology B. The same effect is achieved, if a component is split up into two observation points (C3.1 and C3.2 in Topology C). The calculation of the SC_o for Topology C is also correct and unambiguous. This simple example suggests that topology has a major influence on the calculation of a diagnosis with SFL. The goal of this paper is to study these effects with the help of GA, and derive general characteristics of topologies resulting in high diagnosability.

Table 2: Activity, SC for the topologies in Fig. 1

Topology A

Service	Activity for Topology A	SC_o
S5	00000000000000000000000000000000000000000000	0.000
S1	10001111010000011000010100101100000000100	0.592
S4	10011110111110011011000101011000100100	0.742
S3	11111111111111111111111111111111111111111	1.000
S2	11111111111111111111111111111111111111111	1.000
S0	11111111111111111111111111111111111111111	1.000
Output	11111111111111111111111111111111111111111	

Topology B

Service	Activity for Topology B	SC_o
S5	00000000000000000000000000000000000000000000	0.000
S1	00000000000001000010000000110000010000100	0.340
S4	00111010011000000011101010111001001010001	0.668
S0	11111111111111111111111111111111111111111	0.949
S2	11111011111110111111111111111111111111111	0.973
S3	11111011111110101111111111111011111111111	1.000
Output	11111011111110101111111111111011111111111	

Topology C

Service	Activity for Topology C	SC_o
S5	00000000000000000000000000000000000000000000	0.000
S1	10100000000000000001000000100000000000000	0.344
S4	00010000011000110111001010011100000000	0.445
S2.2	10010010000001110010111101101111001110101	0.601
S0	11111111111111111111111111111111111111111	0.689
S2.1	10011001001110011101101110101100001001 10	0.851
S3	10010001000010111101101110100100011001 10	1.000
Output	10010001000010111101101110100100011001 10	

3 GA for Topology Optimization

We favor genetic algorithms over other optimization heuristics, since they are adequate for our problem domain [4] and easy to apply. GA represent a group of optimization techniques, loosely related to the mechanisms of natural evolution with reproduction and selection [7]. The parameters of the optimization problem are encoded as binary string (chromosome). Each chromosome represents an individual in a pool of solutions (population). During reproduction, pairs of individuals are selected for recombination and some parts of their chromosomes form a new individual. This is termed crossover and controlled by the crossover operator according to probability P_c. After recombination, individual bits of the new chromosome are mutated by a mutation operator according to a low mutation probability P_m. The resulting new individuals are assessed with the fitness function. This measures how well an individual solves the original problem. Fitter individuals have a higher chance to reproduce. This is controlled by the so-called selection operator. The fittest individuals remain in the population and build the basis for the next generation.

In SFL, the topology is represented in the activity matrix. It expresses for every observation point (monitor), whether it has been activated in a transaction or not. The coverage of the topology can be expressed as one binary string, making a mapping to a GA-chromosome straightforward. Every line in the activity matrix becomes a substring of the chromosome.

Table 3: Fitness A: high overall SC

```
# Fitness A: high overall SC
def f_high(chrom, act)
  # genotype -> phenotype transfer
  activity = Array.new
  while (a=chrom.take(act)) != [] do
    activity << a
    chrom = chrom.drop(act)
  end
  # SC calculation
  sc = Array.new
  activity.each do |output_vec|
    activity.each do |activity_vec|
      sc<<ochiai(activity_vec, output_vec)
    end
  end
  # fitness: sum up sc values
  fitness = sc.inject{|sum,x| sum + x}
  return fitness
end
```

The fitness distinguishes good from poor solutions, and it represents the adequacy of a topology to support the calculation of a diagnosis. Diagnosability can be expressed in terms of the extent to which all diagnoses carried out on an activity matrix coming from that topology, are correct diagnoses. In other words, if a topology is organized such that every faulty component can be identified correctly, the topology may be referred to as highly diagnosable. This can be achieved by consecutively setting all components used in the activity matrix to be faulty, and then calculating the similarity coefficient for each fault scenario. This yields a value representing how well a topology facilitates the discovery of faults in components. Topologies leading to higher fitness values will lead to better pinpointing of all faulty components.

The ruby-method `f_high` (Fitness A in Table 3) represents the basic fitness function yielding high overall SC. First, in the so-called *genotype-phenotype transfer*, the GA chromosome is translated into the problem domain, i.e. the binary gene-string is transformed into a binary activity matrix. Second, each component activation vector is set to be the output vector, and the SC is calculated. Third, the SC values are summed up.

4 Experiments

We performed a number of experiments in order to have GA generate highly diagnosable topologies, and then to derive general characteristics for diagnosable topologies. The genetic algorithm used for these experiments can be downloaded.[1] It uses the following rudimentary operators.

Two individuals are selected for recombination based on tournament selection [17]. This chooses N_t individuals from the population randomly, and returns the fittest in this tournament. The actual recombination is done according to the uniform crossover operator [22]. It determines for every bit in the chromosome, according to a probability P_c, whether the value for the new individual (offspring) is taken from the first or from the second parent.

The other GA-parameters depend on the complexity of the particular problem size to be solved. The population size N_p, and the tournament size N_t are set to different values in the different experiments, reflecting the chromosome size of the respective problem, i.e. according to the size of the activity matrix (or based on experience). Bigger activity matrices represent larger search spaces and require bigger populations for better sampling of the search space. Experiments with large topologies are possible but would require more space for presentation. Therefore, the topologies shown are limited to five components. Experiments with larger numbers of components yield similar results. The GA maintains and evolves the N_p fittest individuals. Crossover probability P_c is set to 0.5 in all experiments, and mutation probability P_m is set to a low value of 0.001. These were determined through initial experiments and found to provide acceptable results. Every experiment was repeated 20 times. There may be better GA implementations or operators to chose from, however, the ones introduced here are sufficient to produce usable results.

Assessing the Setup.

The first experiments performed serve as assessment in terms of whether or to which extent the GA is able to generate highly diagnosable activity matrices. We assume a diagnosable topology is represented by high overall SC values. This can be tested by iteratively setting the output vector in the fitness function equal to each component's activation vector (Fitness A in Table 3). Each component is set to be faulty in the calculation of the SC (single fault case), resulting in $SC_o = 1$ for this comparison, and we expect the GA to produce activity matrices in which all component activations are alike. An example is shown in Table 4, above. The first activity matrix (fitness=16.75) represents the best random individual from the first generation. The second activity matrix (fitness=24.88) represents the fittest individual after 200 generations. The success of this optimization example is quite obvious. All component activity vectors are highly similar, representing a highly diagnosable activity matrix expressed by the calculation of high overall SC. In fact, the most optimal solution in this example is fitness=25, when all combinations of component activity vector and output vector yield a 1.0 as SC value, i.e., when they are identical. In this example, the fittest individual is only 1 bit flip away from the optimal solution, i.e. in the penultimate spectrum of C_1.

Even though, this experiment is successful in terms of assessing our experimental setup, it is useless in diagnosis, because the activity matrix represents a topology in which all components are tightly coupled. If C_1 is invoked, all other components will also always be invoked, leading to components C_1 to C_5 being assigned the same ranking (SC = 1.0; compare with Topology A in Table 2), and resulting in an ambiguous diagnosis. As a consequence, we have to extend the adequacy criterion for topologies: "A topology is diagnosable, if it facilitates the detection of all faults in a system, *and their unambiguous identification*," i.e. it must not generate duplicate top SC_o.

Topologies for Discriminable Diagnoses.

In this experiment, the fitness function from the previous setup is adjusted to award topologies higher fitness, which result in high overall SC, but also lead to *discriminable* diagnoses, thereby addressing ambiguity. The fitness function `f_discrim` (Fitness B in Table 5) illus-

[1] https://github.com/SERG-Delft/rusiga

Table 4: Assessment of the experimental setup

100 Generations, 40 Activations	
N_p=120, N_t=6, P_c=0.5, P_m=0.001	
best random individual (fitness=16.75)	
C_1	10101010011110010111110001101011101000100
C_2	00111011001100001011010101011001111010 1
C_3	10010100110111111100001111011000101111011
C_4	01011011100110011001011010111000101100 01
C_5	01011111110010010101010011101010101001
best final individual (fitness=24.88)	
C_1	01111011101111111011111011111111101101010 1
C_2	01111011101111111011111011111111101101111
C_3	01111011101111111011111011111111101101111
C_4	01111011101111111011111011111111101101111
C_5	01111011101111111011111011111111101101111

trates this extension. It awards individuals that lead to one top ranked component, and a number of lower-ranked components. Moreover, it can be configured to minimize (`diff=:low`) or maximize (`diff=:high`) the difference between the top ranked and all lower-ranked components. Table 6 shows examples for both optimization goals.

Table 5: Fitness B: discriminable SC

```
# Fitness B: discriminable SC
def f_discrim(chrom, act, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation
  # same as f_high()
  ...
  # fitness: discriminiable SC
  highest_sc = (sc.sort!)[-1]
  pivot = sc.find_index(highest_sc)
  low_sc = sc[0..pivot-1]
  top_sc = sc[pivot..-1]
  sum_top = top_sc.inject {|sum,x| sum+x}
  sum_low = low_sc.inject {|sum,x| sum+x}
  return sum_top - sum_low if diff==:high
  return sum_low - sum_top if diff==:low
end
```

Adjusting `diff` to `:high` leads to a large number of '0's in the final activity matrix compared to a random activity matrix from the early generations, representing a lot of unique component activation. This means that discriminable diagnoses, indeed, can be supported by the topology of the system, and that inactivity of the components, indicated through the many zeroes, supports this. In other words, high diagnosability can be achieved through inactivity observations, or through activation of components in isolation, which is the opposite of tight component coupling. This is an interesting result, because for the topology it means, that having components which may be activated individually rather than in combination with other components, helps separating system executions, and thus, improves the diagnosability of the system. This comes from how the SC_o calculates similarity. Completely inactive spectra are ignored by the SC_o, but spectra with fewer activations provide more useful information for SFL than spectra with more activations. For example, a spectrum with $a_i = [0, 0, 0, 0, 1]$ is more useful than another one with $a_j = [1, 1, 1, 1, 0]$, because if the transaction a_i fails, this will result in the only one activated component in a_i being blamed more. This outcome may seem like "the bleeding obvious," but, because complete decoupling of all components is not realistic in real systems, in the future, we will have to assess whether or to which extent a GA may be able to generate optimal monitoring locations that help to exploit this property, at least to a certain extent.

Table 6: Examples for discriminable diagnoses

30 Generations, 40 Activations	
N_p=50, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**	
fitness=-2.98 (best random individual)	
C_1	11101000100010001011001110011101001000 11
C_2	10001010001001001001101000000000011001110
C_3	11110100001001011000010001001001011100 00
C_4	10010011111101111001111101110110000101 00
C_5	01001101110000100101101100110100000101001
fitness=4.606 (best individual after 30 gen.)	
C_1	00001000000010000000000011000000111000 10
C_2	01100000000001000000101000010001000011 00
C_3	00000000010000100010000000100000000000 01
C_4	00010011001000010001000000000000000010000
C_5	11000101100100001100010100001110000000000

30 Generations, 40 Activations	
N_p=50, N_t=3, P_c=0.5, P_m=0.001, **diff=:low**	
fitness=7.092 (best random individual)	
C_1	10011111010100001000111011111001000111 01
C_2	10110110000101101111001111011101100110 10
C_3	11001110010101101011001011111110101011 10
C_4	01011011100010001010101011011010011 00011
C_5	10100010010001101011000110011110110101 1
fitness=12.978 (best individual after 30 gen.)	
C_1	10111110010111111000101111111001111111 11
C_2	11111110110111111011101111111001111110 11
C_3	11111110110111111010001111111111111111 01 1
C_4	01111010110111001010101111111001011111 11
C_5	11111000110111111010001111111001111110 11

Setting `diff` to `:low` shows different results. Even though the activity matrix contains many '1's, indicating tight coupling between the components, conclusive diagnoses can be calculated, if the topology can provide just enough discriminative information, e.g. some '0's in some spectra. Looking only at the failing spectra in which each component was activated, would lead to ambiguous diagnoses (comparable with Topology A in Table 2). Because there is slight variation in other spectra to compensate for the tight coupling, the information contained in the activity matrix is just diverse enough in order for the diagnosis algorithm to come up with an unambiguous ranking. An increase in observation diversity can be achieved by adding observation points. One approach could be the inclusion of observations representing the invocation links between the components. Another approach is the instrumentation of the components themselves in order to acquire more diverse observations. This second approach has been demonstrated to improve diagnosis considerably for service-based systems [6]. In any case, both approaches also raise the question of the optimal number of observation points for high diagnosability w.r.t. low monitoring overhead, to be addressed in future work.

Topologies for Intermittent Fault Behavior. In the previous experiments, activation of a faulty component always lead to a failure. Here, we would like to assess to which extent topology influences the quality of the diagnosis when components exhibit intermittent fault behavior. Intermittency, i.e. a component fails occasionally, is quite common in software, and it is not attributable to random faults (as in hardware). Even though, software exhibits deterministic fault behavior, intermittency comes from the mismatch between the monitoring granularity and the activation granularity (basic block level). Hence, intermittency presents a monitoring topology issue.

Fitness function `f_randinterm` (Fitness C in Table 7) realizes intermittency through removing all '1's from each output vector except for a number of randomly chosen ones (e.g. 3 random failure observations). This

Table 7: Fitness C: random intermittency and Fitness D: constant intermittency

```
# Fitness C: random intermittency
def f_randinterm(chrom, activ, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation
  sc = Array.new
  activity.each do |output_vec|
    output_vec.remove_all_ones_except_rand(3)
    activity.each do |activity_vec|
      sc<<ochiai(activity_vec,output_vec)
    end
  end
  # fitness: discriminable
  # same as f_discrim()
  ...
end
# Fitness D: constant intermittency
def f_constinterm(chrom, activ, diff=:high)
  # genotype -> phenotype transfer
  # same as f_high()
  ...
  # SC calculation with const. output vector
  output_vec = [0,0,0,1,0,0,0,0,0,1,0,0,0,...]
  activity.each do |activity_vec|
    sc<<ochiai(activity_vec,output_vec)
  end
  # fitness: discriminable
  # same as f_discrim()
  ...
end
```

yields similar results as presented in Table 6, with `diff` set to `:high` and `:low`, respectively, so we omitted an example. Consecutively using each activation vector as output vector, leads the optimization to be focused only on the generation of high/low differences between top ranked activations and the lower ranked activations, thereby ignoring the intermittency target.

Amending the fitness function by focusing on only one faulty component, leads to a more differentiated outcome (through Fitness D, in Table 7). Table 8 shows two examples with five constant failures seeded into the output vector, and with `diff` set to `:high` and `:low`, respectively. Looking at the two examples, the solution of the GA to the intermittency problem is both cunning and ironic: *"in an optimal topology, faulty components should only be executed when they are guaranteed to fail,"* which avoids intermittency altogether and is not very useful. Further, when `diff` is set to `:high`, it becomes apparent that when the failing component, C_5 in this example, is activated, none of the other components is activated, suggesting again, that the ability to activate components in isolation is advantageous. And when `diff` is set to `:low`, ambiguous diagnoses can be resolved through additional observations, i.e. through the very few additional '1's in the bottom activity matrix. This confirms our previous observations. Intermittency cannot be addressed with this kind of experiment.

Freely Evolved Topologies.

Up to this point, we have had the GA evolve topologies based on a predefined output vector with seeded faults. That way, we could define the interesting error scenarios, and have the GA generate optimal activity matrices. In this experiment, we let the GA not only evolve the activity matrices, but also their corresponding output vectors. It means, we have no control over the number of failure observations generated in the output vector, and we cannot tell whether the diagnosis is correct, because we cannot seed any particular faults. For these experiments, a 6th component is added to the GA chromosome representing the output vector, and Fitness E in Table 9 is used for evaluation of the individuals. The fitness function is slightly different compared to the earlier ones, because of the output vector taking part in the evolution. Setting `diff` to `:low` results in a selective pressure favoring many failure observations to be produced as shown in the example activity matrix on the top right hand side of Table 10. Because the number of failing transactions is unrealistically high for real software systems, we set `diff` to `:high`, resulting in much lower number of failure observations. This is shown in the example activity matrix on the bottom right hand side of Table 10.

Table 8: Examples for fault intermittency

200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**		
	fitness=0.377 (best random individual)	
C_1	00001100111001101000000010001011001100110001	
C_2	00010111110001000010010101110111010101111	
C_3	11001010101011101101010011010000101100001	
C_4	10011001101111001011000100000110111111110	
C_5	10100100101101111101110000100110011111001	
O	01110000000000000000000000000000000000110	
	fitness=1.0 (best individual after 200 gen.)	SC_o
C_1	00001011110000001101001101111100011000	0.00
C_2	10000101010100011010010001111101000100	0.00
C_3	00000100111110111011010001100011010001	0.00
C_4	00001111111100100110001011100000011111001	0.00
C_5	01110000000000000000000000000000000000110	1.00
O	01110000000000000000000000000000000000110	

200 Generations, 40 Activations N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**		
	fitness=1.105 (best random individual)	
C_1	11111100001100010001111110001001111001 0	
C_2	11111001000100010001111000011000000100 11	
C_3	01111001111010101011010001100000111101 01	
C_4	01111101001000111010001110001111000001 01	
C_5	01110100100101001001001010100011111101 10	
O	01110000000000000000000000000000000000110	
	fitness=2.652 (best individual after 200 gen.)	SC_o
C_1	01110000000010000000000000000000000000110	0.91
C_2	01111000000000000000000000000000000000110	0.91
C_3	01110000010000000000000000000000000000110	0.91
C_4	01110000000000000000000000000000000000110	1.00
C_5	01110000100000000000000000000000000000110	0.91
O	01110000000000000000000000000000000000110	

Table 9: Fitness E: Freely evolved topologies

```
# Fitness E: freely evolved
# with output vector
def f_discrout(chrom, act, diff=:low)
  # genotype -> phenotype transfer
  # same as f_discrim()
  ...
  # SC calculation
  # ouput -> last comp act vector
  output = activity_matrix[-1]
  activity_matrix.delete_at(-1)
  sc = Array.new
  activity_matrix.each do |activ|
    sc << ochiai(activ, output)
  end
  # fitness: discriminable SC
  top_sc  = (sc.sort!)[-1]
  top_cnt = sc.count(top_sc)
  low_sc  = sc[0..-2]
  sum_low = low_sc.inject {|sum,x| sum+x}
  return (top_sc - sum_low) / top_cnt if diff==:low
  return (sum_low - top_sc) / top_cnt if diff==:high
  #return (sum_low - top_sc) / (top_cnt + output.count(1))
  # favor. low num. of failures
end
```

Two noteworthy results can be observed in this second case. First, as noted earlier, being able to activate components individually supports the diagnosis. Second, intermittency can be dealt with. The faulty

Table 10: Examples for Freely evolved topologies

200 Generations, 40 Activations		
N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**		
C_1	11111111111111111111111111111101111111	0.97
C_2	10111111111111111011111111111111111111	0.98
C_3	10111111111111111111111111011111111111	0.98
C_4	10111111111111111111111111111111111111	1.00
C_5	10111111111111111111111111111101111111	0.98
O	10111111111111111111111111111111111111	
200 Generations, 40 Activations		
N_p=200, N_t=3, P_c=0.5, P_m=0.001; **diff=:high**		
C_1	11000111000000011001010110001010100100000	0.00
C_2	01000000011010000000100000101011100000001	0.00
C_3	01011000010001000010001000100100011000000	0.80
C_4	01110000010000100100000100010000010010111	0.00
C_5	10000010011000011101000011100010100111000	0.00
O	00001000000001000010001000000100011000000	

component, C_3 in this example, is sometimes invoked without resulting in a failed transaction. This leads to the $SC_o < 1.0$. In fact, when C_3 is invoked in isolation, it fails, if it is invoked in combination with any of the other components it passes. This is a clear indicator of a missing observation point, either in the component C_3 itself, or in one of its peers which is invoking it individually. Translated to a real system it means, that if $C3$ is invoked from an external (unmonitored) system, it is exercising a different internal route, than if it was called from one of its monitored peers. In future work, we will assess to which extent missing observation points can inhibit proper diagnosis.

Freely Evolved Topologies with Fewer Failures. Even when `diff` is set to `:low`, we can get useful results, when the fitness function favors individuals with low number of failures. We can amend the Fitness E shown in Table 9 by adding a scaling factor, i.e. `output.count(1)`, shown in the last line of the fitness function's code, penalizing individuals with high numbers of failures. Table 11 shows an example with a an activity matrix containing only one failure.

Remarkable, again, is the high number of zeroes indicating that monitoring of inactivity is advantageous for high diagnosability. This is in line with our earlier results. However, the result shown in Table 11 also hints to another interesting topological issue. Even though, all components are activated together in case of the failing transaction, representing tight interaction of the components in the fault case, the activity matrix contains enough discriminative information in order to reach an unambiguous diagnosis. This is, again, a strong indicator that the ability of a monitoring topology to observe various combinations or patterns of component invocations will help in reaching unambiguous diagnoses. The other occasional activations lead to sufficiently diverse information in order to being able to separate the tight coupling of the components on failure.

From this observation, we can deduce that not only diverse coverage benefits diagnosability, but moreover, also distinct coverage. In other words, topologies with more diverse execution routes, covering distinct components, facilitate diagnosability. In the topology, this can be achieved through monitoring not only activation or non-activation of a particular entity, i.e. the fact that something has been used, but also through monitoring the context in which something has been used, i.e. incoming and outgoing combinations of activations. In other words, the fact that something has been covered through various routes, or in particular sequences, which can be monitored, has an influence on the diagnosability of a topology. In the future, we will take a closer look at the influence of the traces leading to an activity matrix, rather than merely the activations themselves.

Table 11: Freely evolved topologies, fewer faults

200 Generations, 40 Activations		
N_p=100, N_t=3, P_c=0.5, P_m=0.001; **diff=:low**		
	fitness=0.081 (best random individual)	
C_1	00110101001110011111111110010100010001111	
C_2	00111101010010110001001110100101011101111	
C_3	00111111000011001111000011100001001101111	
C_4	01100100100011110100000110011000000100111	
C_5	01101011010000010001001101101100011110001	
O	00101110001101110000001001010100011011111	
	fitness=0.44 (best indiv. after 200 gen.)	SC_o
C_1	00101001001001000000011000001000001100000	0.316
C_2	00001110101011000011001000000100000100000	0.289
C_3	00001010000100111011000000000000010000110	0.302
C_4	00101000011000000000000111000000011000011	0.302
C_5	01101000100000100010010001100000001000010	0.302
O	00001000000000000000000000000000000000000	

5 Discussion

5.1 Revisiting the Research Questions

In the introduction, we asked ourselves *how genetic algorithms can be used to study the effects of the monitoring topology on the diagnosability of systems.* We will now address this problem by providing answers to the two research questions formulated:

RQ1: How can search-heuristics such as genetic algorithms be used to determine better diagnosable topologies? The topology of a system is represented by an activity matrix, whereby, for each observation point, it expresses whether that point has been activated. The coverage of all observation points can be expressed as a binary string, making a mapping to a GA-chromosome straightforward. For the fitness, we propose several approaches. First, a function that expresses the diagnosability of a monitoring topology, i.e., the extent to which all diagnoses carried out on an activity matrix coming from that topology, are correct diagnoses. In the fitness function, each component is set to be faulty per diagnosis. Then, the fitness function calculates to which extent all similarity coefficients combined from all runs represent correct and distinguishable diagnoses. This yields a value representing how well a topology facilitates the discovery of each potential fault in every component. This basic fitness function can be amend in order to address the different optimization criteria required in the different experiments, e.g. favor high or low differences in the SC_o, or favor output vectors with low number of failures.

RQ2: What are characteristics of topologies that are better diagnosable? According to the fitness function, a topology is a "good" or a diagnosable topology, if it facilitates the detection of all faults in a system in an unambiguous manner. The application of GA hints at a number of routes to satisfying this fitness goal. Our results show that

- being able to invoke components in isolation is beneficial for diagnosability, because it helps separate component involvement in system executions better.

- adding observation points (monitors) in the system, and including the monitoring of inactivity, helps separating system executions, which also facilitates the diagnosability of the system.
- including monitoring of the system context (external components from other systems, incoming and outgoing activations) can support diagnosability through incorporating different invocation routes.
- including tracing information which represents combinations or distinct patterns of component coverage, may suport SFL-based diagnosis.

All these items also raise the question of the optimal number of observation points for high diagnosability w.r.t. low monitoring overhead.

5.2 Lessons Learned

Besides the more general characteristics of diagnosable topologies stated above, the application of GA taught us a lot about the behavior of the SFL approach. It is interesting to see how a search heuristic cannot only help to provide solutions, but also point to issues, both known, and unknown.

In the initial assessment of our setup, the GA generated topologies with tightly coupled components. This was due to our poor fitness definition. We knew already that tight component interaction is bad for diagnosability of a topology, and the GA was, in fact, pointing to this issue, so that in subsequent experiments, the fitness function could be adjusted.

The fact that having fewer activations within a spectrum provides better information for SFL than more activations was not obvious initially. Creating monitoring topologies that lead to such observations is, therefore, an essential goal for future work.

Finally, from the last experiments we can deduce that the context of activity is an important factor in the calculation of a diagnosis. In other words, if a component is activated, which route did this activation take? We knew already that introducing more information into the calculation of the SC yields better diagnoses. But this points to very particular information to be included, i.e., the activation paths through the system. In future work, we will derive the activation sequences from the traces generated by the monitors and encode this in the activity matrix.

5.3 Threats to Validity

In this initial application of GA to studying the effects of topologies on diagnosability, we have used activity matrices instead of real topologies. An activity matrix represents component involvement in system transactions and must be regarded as a simplification of a topology. It does not explicitly express the links between components. We can, therefore, only infer very general characteristics of potentially diagnosable topologies.

In the experiments, we have only looked at a low number of observation points (monitors), and at a low number of observations (spectra). We are aware of the fact that the number of observations and observation points affect the achievable results, but we decided to treat the generation of variable numbers of observation points as a problem in its own right, to be addressed in the future.

6 Related Work

Literature describing the application of genetic algorithms to the optimization of topologies is abundant. For instance, Kumar et al. [14] propose a general approach based on GA to design network topologies for distributed systems, in order to achieve network reliability; Madeira et al. [16] develop a computational model to optimize topologies of linear elastic structures with GA; the authors of [9] use GA to optimize the topology of hardware circuit against parallel flows.

In software engineering, the authors of [10], [19], and [20] propose to apply multi-objective GA to automatically synthesize software architectures. The architectural patterns are used for mutations and the quality metrics are used as fitness function to assess each architecture. Their research results conclude that their approach of architecture synthesis based on GA is able to produce a set of reasonable architectural solutions. However, only two quality attributes, i.e. modifiability and efficiency, were considered in their approach to generate software architectures. Lutz [15] use metaheuristics to evolve good hierarchical decompositions. Decomposition is related to our problem of placing monitors at strategically optimal locations.

Harman [11] states that "metrics are fitness functions too". We acknowledge this by defining fitness functions for diagnosability. Kim and Park [13] propose the application of reinforcement learning in self-managing systems. In particular, they mention software architecture. Our approaches are intended to contribute to self-adaptive and self-managing systems.

Piel et al. [18] apply spectrum-based fault localization techniques together with online monitoring to recover health information and pinpoint problematic components for self-adaptive systems. Abreu et. al. [3] present a diagnosis approach combining spectrum-based fault localization and model-based diagnosis techniques, which is able to locate multiple faulty components with relatively low cost.

7 Summary, Conclusions and Future Work

In this paper, we outlined how genetic algorithms can be used to study the effects of monitoring topologies on SFL-based diagnoses. We defined a simple one-to-one mapping between the chromosome of a genetic algorithm and an activity matrix to be used by SFL, plus several fitness functions representing diagnosability. Activity matrices were used as simplifying models for real topologies. Explorative experiments revealed a number of general characteristics of topologies that support diagnosability, and we learned to better understand how topology affects the calculation of diagnoses.

The vision of our research is that, eventually, we would like to be able to have a search heuristic generate the most optimal monitoring topology in terms of high diagnosability for any arbitrary existing system. In the future, therefore, we will have to look at how real topologies can be encoded for GA, instead of

merely using activity matrices representing topologies. This can be done either with the help of a topology simulator[2], or with real systems. Other issues to be addressed in the future are the inclusion of context information (derived from traces) in the calculation of the diagnosis, and the inclusion of more monitors. This last aspect represents a multi-objective optimization problem in its own right, i.e. generate topologies for optimal diagnosability with minimal monitoring overhead.

Acknowledgements: Thanks to NWO for sponsorship, and our industrial partners Adyen and Exact.

References

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] R. Abreu, P. Zoeteweij, and A. J. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proc. Int'l Symp. on Dependable Computing (PRDC)*, pages 39–46. IEEE, 2006.

[3] R. Abreu, P. Zoeteweij, and A. J. van Gemund. Spectrum-based multiple fault localization. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*, pages 88–99. IEEE, 2009.

[4] C. Chapman, K. Saitou, and M. Jakiela. Genetic algorithms as an approach to configuration and topology design. *Mech. Des.*, 116(4):1005–1012, December 1994.

[5] C. Chen, H.-G. Gross, and A. Zaidman. Spectrum-based fault diagnosis for service-oriented software systems. In *Proc. of the Int'l Conf. on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012.

[6] C. Chen, H.-G. Gross, and A. Zaidman. Improving service diagnosis through increased monitoring granularity. In *7th Intl Conf. on Software Security and Reliability*, page to appear, Washington, DC, June, 18–20 2013.

[7] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, 1989.

[8] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. van Gemund. Spectrum-based sequential diagnosis. In *Proc. Int'l Conf. on Artificial Intelligence (AAAI)*, pages 189–196. AAAI Press, 2011.

[9] G. Granelli, M. Montagna, F. Zanellini, P. Bresesti, and R. Vailati. A genetic algorithm-based procedure to optimize system topology against parallel flows. *Power Systems, IEEE Transactions on*, 21(1):333–340, 2006.

[10] Hadaytullah, S. Vathsavayi, O. Raiha, and K. Koskimies. Tool support for software architecture design with genetic algorithms. In *Proc. International Conference on Software Engineering Advances (ICSEA)*, pages 359–366. IEEE CS, 2010.

[2] https://github.com/SERG-Delft/sfl-simulator

[11] M. Harman and J. A. Clark. Metrics are fitness functions too. In *Proc. of the Int'l Symp. on Software Metrics (METRICS)*, pages 58–69. IEEE, 2004.

[12] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. In *Proc. SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE)*, pages 83–90. ACM, 1998.

[13] D. Kim and S. Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, volume 0, pages 76–85. IEEE, 2009.

[14] A. Kumar, R. M. Pathak, Y. P. Gupta, and H. R. Parsaei. A genetic algorithm for distributed system topology design. *Comput. Ind. Eng.*, 28(3):659–670, 1995.

[15] R. Lutz. Evolving good hierarchical decompositions of complex systems. *Journal of Systems Architecture*, 47(7):613–634, July 2001.

[16] J. A. Madeira, H. Rodrigues, and H. Pina. Multi-objective optimization of structures topology by genetic algorithms. *Advances in Engineering Software*, 36(1):21–28, 2005.

[17] B. Miller and D. Goldberg. Genetic algorithms, tournament selection and the effects of noise. Technical Report 95006, IlliGAL Report, Dept. General Engineering, University of Illinois at Urbana Campaign, July 1995.

[18] E. Piel, A. Gonzalez-Sanchez, H. Gross, and A. van Gemund. Spectrum-based health monitoring for self-adaptive systems. In *Proc. Int'l Conf. Self-Adaptive and Self-Organizing Systems (SASO)*, pages 99–108. IEEE, 2011.

[19] O. Räihä, K. Koskimies, and E. Mäkinen. Genetic synthesis of software architecture. In *Proc. of the International Conference on Simulated Evolution and Learning (SEAL)*, volume 5361 of *LNCS*, pages 565–574. Springer, 2008.

[20] O. Räihä, K. Koskimies, and E. Mäkinen. Generating software architecture spectrum with multi-objective genetic algorithms. In *Third World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 29–36. IEEE, 2011.

[21] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In *European Softw. Engineering Conf. & Symp. on Foundations of Softw. Engineering (ESEC/FSE)*, volume 1301 of *LNCS*, pages 432–449. Springer, 1997.

[22] G. Syswerda. Uniform crossover in genetic algorithms. In *Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[23] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. van Gemund. Diagnosis of embedded software using program spectra. In *Proc. Int'l Conf. and Workshops on Engineering of Computer-Based Systems (ECBS)*, pages 213–220. IEEE, 2007.

Formal Specification and Synthesis of FDI through an Example

Marco Bozzano and **Alessandro Cimatti** and **Marco Gario** and **Stefano Tonetta**
Fondazione Bruno Kessler, Trento, Italy
e-mail: {bozzano,cimatti,gario,tonettas}@fbk.eu

Abstract

The correct operation of complex critical systems is increasingly relying on the ability to detect whether (and which) faults are occurring. This task, known as Fault Detection and Identification (FDI), is typically carried out by an FDI component that, observing the sequence of values conveyed by some predefined observables, triggers a set of predefined alarms. An effective FDI system can provide vital information to steer the containment and recovery of faults.

We recently proposed a formal framework to support the design of FDI for discrete event systems. The framework is based on a logical language for the specification of FDI requirements and it captures problems such as local diagnosability and maximality, by relying on a semantics based on temporal epistemic logic. This enables the formal verification of the specification and the synthesis of correct-by-construction FDI components.

In this paper, we show the merits of the framework by applying it on a simple example, working out the details of the synthesis algorithm, and by reporting the experience on an industrial case-study from the aerospace domain.

1 Introduction

The correct operation of complex critical systems (e.g. trains, satellites, cars, or industrial plants) increasingly relies on the ability to detect when and which faults occur, since an effective diagnostic system can provide vital information to drive the containment of faults and their recovery. This task, known as Fault Detection and Identification (FDI), is typically carried out by an FDI component that triggers a set of predefined alarms based on the sequence of values conveyed by some predefined observables.

Faults are often not directly observable, thus we can detect their occurrence only by observing the effects they have on the observable parts of the system. Moreover, faults manifest in different ways and they might interact with each other in complex ways. For these reasons, the design of FDI components is an extremely hard task.

The state of the practice lacks a structured and effective methodology to design FDI components. These are often completed late in the system development cycle, by relying on past success stories. The lack of effective validation tools often results in conservative assumptions, so that the overall system features overly simple behaviors. This difficulty is witnessed in the aerospace sector by a recent ITT issued by the European Space Agency [European Space Agency, 2010], which strongly motivated our work, in particular, by stressing the importance of FDI in terms of alarms that are meaningful for performing the recovery of the system.

In a recent work [Bozzano *et al.*, 2013], we proposed a comprehensive framework to support the design of FDI for discrete event systems. The framework is based on a *logical language* for the specification of FDI requirements. The language is based on common patterns for FDI properties, that allow us to specify a wide class of practical requirements, such as alarms corresponding to the occurrence of a given condition in a given preceding time interval. Patterns are formalized in terms of temporal operators (as in Linear Temporal Logic [Pnueli, 1977]) and epistemic operators (such as certainty, or knowledge [Halpern and Vardi, 1989]). The temporal epistemic logic is in turn interpreted over the system being observed.

The approach covers two novel and orthogonal directions, which are important for the specification of FDI requirements. One direction is the ability to express *local diagnosability*, i.e. the diagnosability condition for which the diagnoser is required to satisfy the alarm specification is localized to traces. The other direction is the *maximality* of the diagnoser, that is, the ability of the diagnoser to raise an alarm as soon as and whenever possible. The formal specification languages allows the verification of a given diagnoser with respect to a given specification, and to automatically synthesize a diagnoser from a given specification. We refer to [Bozzano *et al.*, 2013] for a comparison with related work.

Goal of this paper is to present the approach defined in [Bozzano *et al.*, 2013] through the elaboration of a simple case study: the magicbox example. The magicbox is a grid-like structure, where a ball is able to jump from one cell to another according to a predefined pattern. We work-out the details of the specification and of the algorithm for the synthesis of a correct-by-construction FDI component. The synthesis algorithm is based on the exploration of the space of *belief states*, which are annotated with alarms taking into account the temporally extended requirements. In this paper we focus on providing the motivation and an intuitive description of the framework, while we refer the reader to [Bozzano *et al.*, 2013] for a more comprehensive technical description.

The applicability of the framework is also discussed by

presenting preliminary experiences on an industrial case-study coming from aerospace.

This paper is structured as follows. Section 2 provides some introductory background and the magicbox example. Section 3 describes the syntax and semantics of the specification language. In Section 4, we detail the synthesis algorithm. The results of evaluating our implementation of the framework are presented in Section 5. Section 6 concludes the paper with a hint on future work.

2 Background

In the following we use the word *system* to indicate the composition of the plant and the FDI component; we also use *diagnoser* and *FDI component* interchangeably.

Transition Systems

Plants and FDIs are represented with transition systems. A transition system is a tuple $S = \langle V, V_o, W, W_o, I, T\rangle$, where V is the set of state variables, $V_o \subseteq V$ the set of observable state variables; W the set of input variables, $W_o \subseteq W$ the set of observable input variables; I is the initial formula over V, T is the transition formula over V, W, V' (with V' being the next version of the state variables).

A state s is an assignment to the state variables V. We denote with s' the corresponding assignment to V'. An input i is an assignment to the input variables W. The observable part $obs(s)$ of a state s is the projection of s on the subset V_o of observable state variables. The observable part $obs(i)$ of an input i is the projection of i on the subset W_o of observable input variables. Given an assignment a to a set of variables X and $X_1 \subseteq X$, we denote the projection of a over X_1 with $a_{|X_1}$. Thus, $obs(s) = s_{|V_o}$ and $obs(i) = i_{|W_o}$.

A trace of S is a sequence $\pi = s_0, i_1, s_1, i_2, s_2, \ldots$ of states and inputs such that s_0 satisfies I and, for each $k \geq 0$, $\langle s_k, i_{k+1}, s_{k+1}\rangle$ satisfies T. The observable part of π is $obs(\pi) = obs(s_0), obs(i_1), obs(s_1), obs(i_2), obs(s_2), \ldots$.

Synchronous composition

Let $S^1 = \langle V^1, V^1_o, W^1, W^1_o, I^1, T^1\rangle$ and $S^2 = \langle V^2, V^2_o, W^2, W^2_o, I^2, T^2\rangle$ be two transition systems with $\emptyset = (V^1 \setminus V^1_o) \cap V^2 = V^1 \cap (V^2 \setminus V^2_o) = (W^1 \setminus W^1_o) \cap W^2 = W^1 \cap (W^2 \setminus W^2_o)$. We define the synchronous product $S^1 \times S^2$ as the transition system $\langle V^1 \cup V^2, V^1_o \cup V^2_o, W^1 \cup W^2, W^1_o \cup W^2_o, I^1 \wedge I^2, T^1 \wedge T^2\rangle$. Every state s of $S^1 \times S^2$ can be considered as the product $s^1 \times s^2$ such that $s^1 = s_{|V^1}$ is a state of S^1 and $s^2 = s_{|V^2}$ is a state of S^2.

We say that S^1 is compatible with S^2 iff i) for every initial state s^2 of S^2, there exists an initial state s^1 of S^1 such that $s^1_{|V^1_o \cap V^2_o} = s^2_{|V^1_o \cap V^2_o}$ and ii) for every reachable state $s^1 \times s^2$ of $S^1 \times S^2$, for every transition $\langle s^2, i^2, s'^2\rangle$ of S^2, there exists a transition $\langle s^1, i^1, s'^1\rangle$ such that $i^1_{|W^1_o \cap W^2_o} = i^2_{|W^1_o \cap W^2_o}$ and $s'^1_{|V^1_o \cap V^2_o} = s'^2_{|V^1_o \cap V^2_o}$.

Diagnoser

We can now introduce the idea of diagnoser, that is a machine D that synchronizes with observable traces of the plant P. D has a set $\mathcal{A}$ of Boolean alarm variables that are activated in response to the monitoring of P. In other terms, a diagnoser can be seen as a function that maps a sequence of observations into a set of alarms: $obs^* \to 2^{\mathcal{A}}$.

Formally, given a set $\mathcal{A}$ of alarms and a plant transition system $P = \langle V^P, V^P_o, W^P, W^P_o, I^P, T^P\rangle$, a diagnoser is a deterministic transition system $D(\mathcal{A}, P) = \langle V^D, V^D_o, W^D, W^D_o, I^D, T^D\rangle$ such that: $V^P_o \subseteq V^D_o$, $W^P_o \subseteq W^D_o$, $\mathcal{A} \subseteq V^D_o$ and D is compatible with P. When clear from the context, we use D to indicate $D(\mathcal{A}, P)$. Only observable variables can be shared among the two systems and used to perform synchronization. This gives raise to the problem of partial observability: the diagnoser cannot perfectly track the evolution of the original system. This makes the diagnoser synthesis problem hard.

Linear Temporal Logic

Our specification framework is based on Temporal Epistemic Logic, in order to describe reasoning about the knowledge of the diagnoser. We will only use a subset of this logic in this paper, namely Linear Temporal Logic (LTL) with Past operators, to talk about *diagnosis conditions*.

Given a set of propositional variables $\mathcal{P}$, a formula in LTL with Past is defined as

$$\beta ::= p \mid \beta \wedge \beta \mid \neg\beta \mid O\beta \mid Y\beta \mid G\beta \mid F\beta \mid X\beta$$

where $p \in \mathcal{P}$. We define the usual abbreviations of propositional logic $\vee$,$\rightarrow$ and $\leftrightarrow$, moreover introduce two additional abbreviations: $Y^n\beta = YY^{n-1}\beta$ (with $Y^0\beta = \beta$) and $O^{\leq n}\beta = \beta \vee Y\beta \vee \cdots \vee Y^n\beta$.

The semantics of LTL is given in terms of traces, we denote with $\sigma \models \phi$ the fact that the trace σ satisfies the formula ϕ:

- $\sigma, i \models p$ iff $\sigma[i] \models p$
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$
- $\sigma, i \models \phi \wedge \psi$ iff $\sigma, i \models \phi$ and $\sigma, i \models \psi$
- $\sigma, i \models Y\phi$ iff $i > 0$ and $\sigma, i-1 \models \phi$
- $\sigma, i \models O\phi$ iff for some $0 \leq j \leq i$, $\sigma, j \models \phi$
- $\sigma, i \models G\phi$ iff for all $j \geq 0$, $\sigma, j \models \phi$
- $\sigma, i \models F\phi$ iff for some $j \geq i$, $\sigma, j \models \phi$
- $\sigma, i \models X\phi$ iff $\sigma, i+1 \models \phi$

Intuitively the Yesterday and Once operator make it possible to talk about events in the previous time step, or in the whole past history of the system (respectively). Combining these operators, we can create complex scenarios.

Magicbox

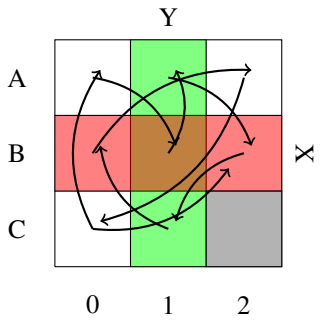


Figure 1: The running example magicbox

A magicbox [Giunchiglia and Ghidini, 1998] is a grid-like structure, in which a ball is able to jump from one cell to another according to a predefined pattern. The movement of the ball is not observable directly, but only through two types of observation points: row and columns. An observer on a row is able to understand when the ball is in its row. However, the observer cannot say anything about the "distance" of the ball, therefore no information on the column

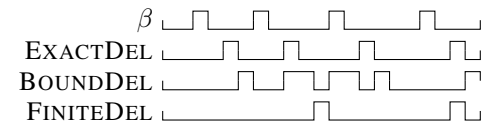


Figure 2: Alarm responses for different delays

can be deduced by this observer. Similarly, the column observers can tell when the ball is in their column, but nothing more. Our running example is the magicbox in Figure 1. This is a 3x3 magicbox, with two observers. The observer x is able to tell when the ball is in the row B, while the observer y is able to tell when the ball is in the column 1. The trajectory of the ball is represented by the arrows, e.g., from A0 the ball will go to B1. To keep the example simple, we blocked the cell C2.

We want to develop a diagnoser that is able to detect the passage of the ball through a certain cell, even if the cell is not observable by itself. Although the ball path is predefined, there is non-determinism in the initial location of the ball and in the possible non-deterministic transitions. For example, from C0 the ball can either go to A0 or to B2.

3 ASL$_K$

The Epistemic Alarm Specification Language (ASL$_K$) makes it possible to specify a set of properties that define when the FDI is correct and complete with regard to a set of alarms, and the level of identification required with respect to a set of faults.

An alarm specification is composed of two parts: the *diagnosis condition* and the *diagnosis delay*. The diagnosis condition is the situation to monitor; this includes a fine-grained distinction between fault detection and fault identification. The diagnosis delay is the allowed delay between the occurrence of the diagnosis condition and its detection; it might be the case that the occurrence of a fault can go undetected for a certain amount of time. Our goal is to define clearly how long this interval can be at most.

Let us assume to have a set $\mathcal{P}$ of propositions representing either faults or elementary conditions for the diagnosis. The set $\mathcal{D}_\mathcal{P}$ of *diagnosis conditions* over $\mathcal{P}$ is an LTL Past formula β.

A diagnosis condition $\beta \in \mathcal{D}_\mathcal{P}$ is a condition of the plant that we want to detect. If β is a fault, the fault must be identified. If β is a disjunction of faults, the detection must be performed, but the identification of the single faults is not required.

An *alarm specification* φ is used to specify how the occurrence of a diagnosis condition β determines the raising of an alarm A. This involves three aspects: the delay, the maximality of the alarm, and the diagnosability of the diagnosis condition. For diagnosability, we will consider two possible values: $diag \in \{local, global\}$; for maximality we consider $max \in \{True, False\}$.

We consider three kinds of delay, that provide the name for the templates in ASL$_K$:

- EXACTDEL$_K(A, \beta, n, diag, max)$
- BOUNDDEL$_K(A, \beta, n, diag, max)$
- FINITEDEL$_K(A, \beta, diag, max)$

Intuitively, the formalization of these template (cf. [Bozzano *et al.*, 2013]) includes two aspects: completeness and correctness. Completeness means that whenever the diagnosis condition is satisfied, the alarm will hold in the future (i.e., no false negative). Correctness means that whenever the alarm holds then the diagnosis condition was met earlier (i.e., no false positives). Specifying the delay provides a more fine-grained way of quantifying the time that an alarm can be silent before considering it as a false negative. In Figure 2, we show the diagnosis condition β in the first row, and the responses for different delay types: exact- (with $n = 2$), bounded- (with $n = 4$) and finite-delay. Lets consider the exact-delay specification: completeness says that whenever the diagnosis condition β is satisfied, the alarm A will hold exactly $n = 2$ steps afterwards; correctness says that whenever the alarm A holds then the diagnosis condition β was met $n = 2$ steps earlier.

If the plant is non-diagnosable for β, then the completeness part of the specification will never be satisfied. This is the classical understanding of diagnosability, and we call this *global* diagnosability [Sampath *et al.*, 1996]. Intuitively, however, it might be possible to contain the non-diagnosability of the system to a subset of the plant. Therefore, we introduce the concept of *local* diagnosability. The idea is to evaluate the diagnosability of a diagnosis condition on single traces instead that on the entire plant. In practice, we require our diagnoser to raise an alarm in all those situations in which there is no uncertainty, given a sequence of observations.

An interesting problem with bounded- and finite-delay specifications is the problem of *maximality*. Lets consider the specification BOUNDDEL$_K(A, \beta, 4, local, False)$. After the occurrence of β a diagnoser can satisfy this specification by raising the alarm at any given time before 4 time-step pass. However, this does not constraint the amount of time the alarm should stay up. Indeed, the alarm could go up, down and up again within the window of 4 time-steps and its behavior would be considered correct. A graphical representation of this situation is depicted in Figure 3, where the first row represents the diagnosis condition, and the second row represents a maximal alarm.

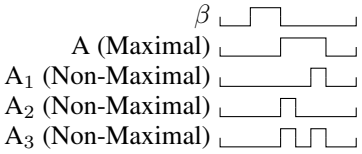


Figure 3: Comparison of non-maximal alarms

Although all the alarms in Figure 3 are correct and complete, some of them introduce an unnecessary delay between the occurrence of the diagnosis condition β and the alarm A; moreover, multiple raising of A could be misinterpreted as multiple occurrences of β.

We address this problem by defining the concept of *maximality*. Intuitively, a diagnoser is maximal if it raises the alarm as soon as possible and keeps it up as much as possible. Adding this requirement makes it possible to specify the unique behavior that a diagnoser must have w.r.t. an alarm specification; therefore, it becomes possible to verify the diagnoser by comparing it against another one.

Example Specification For our magicbox example, we can define several specifications. For example, we could define several diagnosis conditions. The obvious way of do-

ing so is by cell, e.g., $\beta_{C1} = C1$. This could be compared to fault identification, in which we are interested in knowing exactly where the ball is. However, we might loose specificity and say that we want to know if the ball is in one of a given set of cells, e.g., the column A: $\beta_A = A0 \vee A1 \vee A2$. This flexibility makes it possible to specify common problems like fault detection, fault isolation and fault identification.

In ASL_K it is also possible to define temporal properties as diagnosis condition. There are two ways of getting into B2, therefore, we might be interested in knowing whether we arrived from C0 rather than from A1; this can be expressed with the temporal condition $\beta = B2 \wedge YCO$, where Y is the LTL operator "yesterday". The ability of specifying complex scenarios as a diagnosis condition, makes it possible to obtain more informative alarms.

Once we defined the diagnosis conditions, we need to express the acceptable delay, the maximality and type of diagnosability for the alarm. Our example specification consists of the following alarms: $\mathcal{A} = \{$

$$\varphi_1 = \text{EXACTDEL}_K(A(B1), \beta_{B1}, 0, global, True),$$
$$\varphi_2 = \text{EXACTDEL}_K(A(C1), \beta_{C1}, 0, local, True),$$
$$\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)\}$$

In the next section we will see how to build a diagnoser that satisfies the specification $\mathcal{A}$.

4 Synthesis of Diagnoser

In [Bozzano *et al.*, 2013] we present an algorithm that is able to deal with any ASL_K specification, by handling both diagnosable and non-diagnosable systems, and producing a maximal diagnoser. In this section we review the main concepts of the algorithm and apply it to our running example.

The idea of the Belief Explorer algorithm is to generate an automaton that keeps track of the possible states in which the plant could be after the given observations. Similarly to [Schumann, 2004], this translates into generating the power-set of the states of the plant, and define a suitable transition relation among the elements of this set. We call the power-sets defined in this way *belief states*. Each belief state of the automaton can be annotated with the alarms that are satisfied in all the states of the belief state. The automaton, together with the annotations, can be encoded symbolically obtaining the diagnoser.

The diagnoser obtained with this procedure is compatible with the plant, maximal and correct ([Bozzano *et al.*, 2013]).

Belief Automaton

Given a plant $P = \langle V^P, V_o^P, W^P, W_o^P, I^P, T^P \rangle$, let S be the set of states of P. The *belief automaton* is defined as $\mathcal{B}(P) = \langle B, E, B_0, T_b \rangle$ where $B = 2^S$, $E = 2^{W_o^P \cup V_o^P}$ and $B_0 \subseteq B$ and $T_b : (B \times E) \to B$ are defined as follows.

We define $B_0 = \{b \mid$ there exists $u \in 2^{V_o^P}$ such that for all $s \in b$, $s \models I^P$ and $obs(s) = u\}$, meaning that each initial belief state is compatible with one of the possible observations of the system. We implicitly assume that we can initialize the diagnoser by observing the state of the system.

The transition function T_b is defined as follows $b' = T_b(b, e) = \{s' \mid \exists s \in b$ such that $\langle s, i, s' \rangle \models T^P$, $obs(s') = e_{|V_o^P}, obs(i) = e_{|W_o^P}\}$. Intuitively, the belief state b' is a successor of b iff all the states in b' are compatible with the observations from a state in b.

Annotation process

Each of the belief states can now be annotated with the alarms, by checking for entailment of the temporal property $\tau(\varphi)$. To explain this procedure we first consider the simplest case $\varphi = \text{EXACTDEL}(A_\varphi, \psi, 0)$, where $\tau(\varphi) = \psi$ is a propositional formula. We can explore the belief automaton, and annotate with A_φ all the states b that are consistent in the satisfaction of $\tau(\varphi)$:

$$b \models A_\varphi \text{ iff } b \models \tau(\varphi) \text{ iff } \forall s \in b.s \models \tau(\varphi)$$

Note that it might occur that neither $b \models \tau(\varphi)$ nor $b \models \neg\tau(\varphi)$. This is the case if in the belief state there is at least one system state in which $\tau(\varphi)$ holds and one in which it does not; this situation is a witness of *uncertainty* in the belief state, caused by by non-diagnosability.

All other alarms specifications can be reduced to the previous case by performing a pre-processing step to the belief automaton construction. For each alarm specification φ, we add a monitor variable $\overline{\tau}$ in the plant obtaining a plant P', s.t. $P' = P \times (G(\tau \leftrightarrow \overline{\tau}))$, where we abuse notation to indicate the automaton that encodes the monitor variable. Any alarm specification on P can be redefined on P' as $\varphi' = \text{EXACTDEL}(A_\varphi, \overline{\tau}, 0)$, so that $D \times P \models \varphi$ iff $D \times P' \models \varphi'$.

Example

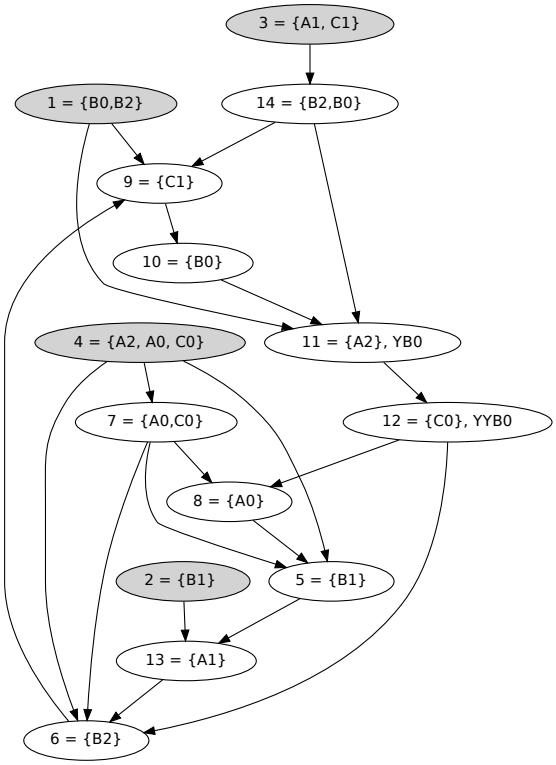


Figure 4: The belief automaton of the running example

Figure 4 shows the belief automaton for our running example, and the specification $\mathcal{A}$ defined in the previous section. To keep the diagram simple, we did not add the observables on the edges. Note that in the diagram state_2 and state_5 have the same belief state but are handled differently because state_2 is an initial state.

At the beginning we do not know where the ball is, therefore our first observation splits the initial belief state in 4 possible belief states (gray nodes) (Figure 5).

X	Y	State
0	0	$4 = \{A2, A0, C0\}$
0	1	$3 = \{A1, C1\}$
1	0	$1 = \{B0, B2\}$
1	1	$2 = \{B1\}$

Figure 5: Initial states observations

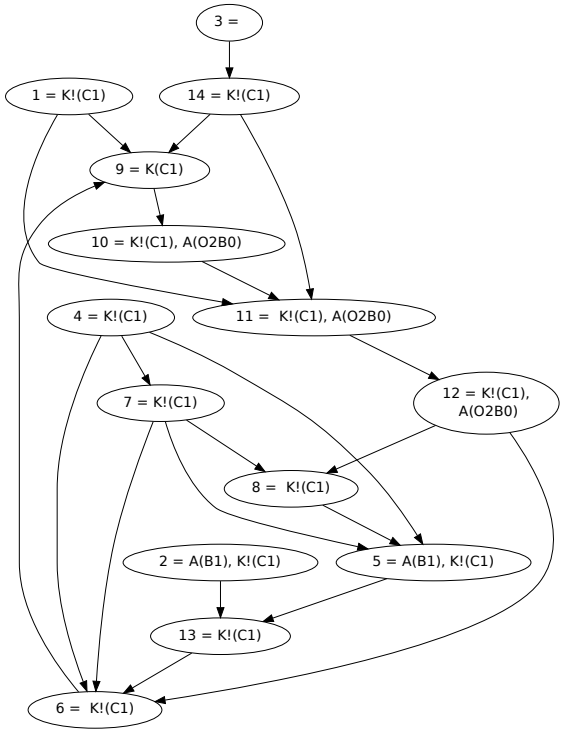


Figure 6: The diagnoser

Lets take the most ambiguous initial state: state_4. This state has transitions to state_5, state_6 and state_7. Each of these transitions will reduce the uncertainty on the location of the ball; in particular, in state_5 and state_6 we will have certainty on the ball location (B1 and B2 respectively). In state_7 we can say for sure that the ball is in the column 0: $\text{state}_7 \models (A0 \vee B0 \vee C0)$.

Finally, note that state_{11} and state_{12} contain the monitor variables $YB0$ and $YYB0$, that will be used for handling the specification $\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)$.

Annotations Once we have constructed the belief automaton, we can navigate it and add the alarm A_φ whenever the temporal formula $\tau(\varphi)$ holds. The result of the annotation process is presented in Figure 6, where for clarity we show only the positive annotations. Lets take the first of our specifications: $\varphi_1 = \text{EXACTDEL}_K(A(B1), \beta_{B1}, 0, global, True)$. We annotate with $A(B1)$ all the states b in which $b \models \tau(\varphi_1) = B1$: state_2 and state_5; all other states are marked with the negation of the alarm $!A(B1)$.

Diagnosability For the second specification $\varphi_2 = \text{EXACTDEL}_K(A(C1), \beta_{C1}, 0, local, True)$ we proceed in a similar way. However, we realize that there is only one state b in which $b \models C1$ holds, i.e., state_9. In state_3 we have an ambiguity between A1 and C1. If our specification were to require global diagnosability, state_3 would be a witness of the non-diagnosability of the system. However, our specification requires local diagnosability. Therefore, we annotate state_9 with $A(C1)$ and all other states with $\neg A(C1)$. It should be clear, that using the same annotation ($\neg A(C1)$) for both state_3 and (e.g.) state_{14} is counter-intuitive. In state_3, we do not know whether the ball is in C1, while in state_{14} we are *sure* that the ball is not in C1. To handle this situation, we break the specification φ_2 in two parts:

$$\varphi_{2+} = \text{EXACTDEL}_K(K(C1), \beta_{C1}, 0, local, True)$$
$$\varphi_{2-} = \text{EXACTDEL}_K(K!(C1), \neg\beta_{C1}, 0, local, True)$$

Intuitively, we express the epistemic operator in the alarm name, therefore encoding all situations of interest: the diagnoser *Knows* that C1 holds, or it *Knows* that it does not. We can now annotate state_9 with $K(C1)$, all the other states (except state_3) with $K!(C1)$ and then complete the annotations with $!K(C1)$ and $!K!(C1)$. After performing the annotation, in state_3 both the alarms $K(C1)$ and $K!(C1)$ will not hold, due to the non-diagnosability. This is particularly useful, because now we can react accordingly; for example, a recovery module might act in a different way knowing whether there is uncertainty on the occurrence of $C1$.

Splitting the specification in a positive and negative part is even more interesting for bounded specification. Lets consider $\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)$. The diagnosis condition is diagnosable with delay 2. We have uncertainty in state_{14}, but then in state_{11} we know that $KO^{\leq 2}B0$ holds (due to the monitoring variable $YB0$). Therefore, we could use the global diagnosability version of this specification. However, it might be interesting to have both positive and negative alarms, in order to annotate state_{14} and state_1 with the information that $O^{\leq 2}B0$ *might* hold there, and distinguish them from all other states in which we are sure that it does not hold (e.g., state_5).

Another situation in which local diagnosability is needed, are the initial states. For example, we have uncertainty for $C1$ in state_3, however, this is the only situation in which we have uncertainty on $C1$. Excluding this state would give us diagnosability, and this is exactly the idea behind local diagnosability.

Maximality In section 3 we introduced the idea of maximality, to express the completeness w.r.t. what the diagnoser could know given a series of observations. φ_3 is a good candidate to explain this concept more in detail.

In Figure 6 we show a maximal annotation. The annotation $A(O2B0)$ is given to state_{10}, state_{11} and state_{12}. This means that, for each state in which $\tau(\varphi) = O^{\leq 2}\beta_{B0}$ holds, we have the annotation $A(O2B0)$. We could argue that having the alarm $A(O2B0)$ only in state_{11} would be correct too. Even more, we would still be correct if we had the alarm on state_{10} and state_{12} only. The three formalizations satisfy our definition of non-maximal bounded-delay specification (cf. [Bozzano *et al.*, 2013]); however, this situation leaves many details on the behaviour of the diagnoser underspecified, making it hard to use the information provided by the diagnoser to devise (e.g.) a recovery or containment procedures. Therefore, maximality is useful to provide a clear and un-ambiguous specification of the diagnoser, and it enables the verification of a diagnoser by comparison of the outputs with another diagnoser.

5 Experimental Evaluation

The algorithm presented in this paper has been implemented on top of the NuSMV model-checker, making it possible to specify and synthesize a diagnoser. The belief explorer synthesis procedure has been implemented by using BDD's [Bryant, 1986]. BDDs make it easy to identify already visited belief states; moreover, having a symbolic characterization of the belief-state makes it possible to check for entailment of an alarm in a belief-states ($b \models \tau(\varphi)$) with a single operation.

To show the feasibility of our approach, we present in this section an experimental evaluation conducted on magic boxes, and discuss the application of these techniques to an industrial case study.

Magicbox Benchmark

In our experimental evaluation we generated random magicboxes by considering the following variables: size of the magicbox $N \in [2, 50]$; number of observables ($M = \lfloor N * k \rfloor$), with $k \in [0.1, 0.9]$ and number of specifications $S \in [1, 20]$.

For several combinations of (N,M,S) we measured the runtime of the belief explorer, and we noticed the (expected) exponential growth w.r.t. the size of the model. Moreover, we noticed how the percentage of observables influences the runtime and memory usage: few observables give rise to huge belief states, where the uncertainty is substantial; many observables, instead, will make it possible to quickly reduce the uncertainty and generate less and smaller belief states. Finally, adding monitoring variables to the system to does not seem to impact significantly the performances.

AUTOGEF

This framework has been used as the base for the European Space Agency AUTOGEF project [European Space Agency, 2010].

The main goal of the project was to explore the possibility of defining requirements for an on-board Fault Detection, Identification and Recovery component (FDIR) and perform the synthesis of an FDIR satisfying the requirements. The problem was tackled by synthesizing the Fault Detection (FDI) and Fault Recovery (FR) separately. The FDI needs to provide enough information to the FR to act accordingly. A case-study for evaluating the approach was defined and evaluated by Thales Alenia Space based on the EXOMARS Trace Gas Orbiter.

The model and the requirements were formalized within AUTOGEF, the FDIR component generated. The system is composed by 11 components with 10 possible faults in total. The generated FD had 754 states. Crucial for the synthesis of the diagnoser was the ability of defining locally diagnosable alarms, since most of the modeled faults were not globally diagnosable.

The correctness of the synthesized FDIR is guaranteed by construction, however, we are currently working in using epistemic model checking techniques to be able to verify the correctness of any given FDI module against the specification.

6 Conclusions and Future Work

Goal of this paper is to illustrate the features of the formal framework for the specification, verification, and synthesis of Fault Detection and Identification components [Bozzano *et al.*, 2013]. We explored the application of the framework on a toy example, and worked out the details of the synthesis algorithm. Finally, we discussed the applicability of the framework also presenting our preliminary experience on an industrial case-study coming from aerospace.

In the future, we plan to extend the framework to deal with asynchronous and infinite-state (timed/hybrid) systems, and with the specification and synthesis of distributed and hierarchical FDI components.

Acknowledgments We would like to thank Régis de Ferluc for providing us the case-study for AUTOGEF.

References

[Bozzano *et al.*, 2013] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. A formal framework for the specification, verification and synthesis of fdi components. In *Technical Report*, 2013. Available at URL https://es.fbk.eu/people/gario/TR-FBK-ES-13a.pdf.

[Bryant, 1986] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Trans. on*, 100(8):677–691, 1986.

[European Space Agency, 2010] European Space Agency. ITT AO/1-6570/10/NL/LvH "Dependability Design Approach for Critical Flight Software". Technical report, 2010.

[Giunchiglia and Ghidini, 1998] F. Giunchiglia and C. Ghidini. Local models semantics, or contextual reasoning = locality + compatibility. In *KR'98*, pages 282–291. Morgan Kaufmann, 1998.

[Halpern and Vardi, 1989] J.Y. Halpern and M.Y Vardi. The complexity of reasoning about knowledge and time. i. lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[Sampath *et al.*, 1996] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. on Control Systems Technology*, 4(2):105–124, 1996.

[Schumann, 2004] A. Schumann. Diagnosis of discrete-event systems using binary decision diagrams. *DX*, pages 197–202, 2004.

Minimal Sequential Diagnosis of Discrete-Event Systems *

Xiangfu Zhao[1] and **Luca Ceriani**[2]
[1]Department of Computer Science, Zhejiang Normal University, Jinhua, China
[2]Department of Information Engineering, University of Brescia, Brescia, Italy
e-mail: xiangfuzhao@gmail.com, luca.ceriani@ing.unibs.it

Abstract

In some discrete-event systems (DESs), the order of occurrence of fault events is very important: for example, aging in power lines may cause fire, and, conversely, fire may cause the failure of power lines. Therefore, in diagnosis of DESs it could be also useful to consider the *order* of faulty events, not only the *set* of faulty events as usually proposed in literature. In addition, candidates encompassing a fewer number of faults are more valuable than those including a larger number of faults. This paper surveys the diagnosis approach proposed by [Grastien *et al.*, 2011; 2012] based on the exploration of the hypothesis space to represent and derive all minimal sequential diagnoses of DESs. We formalize the concept of SE-tree (Sequence Enumeration tree), which is implicit in the cited works and we propose some pruning rules and heuristics to reduce the size of the explored hypothesis candidate space.

1 Introduction

Some disastrous accidents, such as crashed space shuttles, nuclear explosions, and large blackouts, have greatly boosted the research on model-based diagnosis, which processes a model of the system to be diagnosed, where such a model is drawn from first principles [Reiter, 1987], not from experts' knowledge, which can hardly be obtained for new high-tech devices.

Model-based diagnosis of discrete-event systems (DESs) has attracted considerable attention from the middle 1990s [Sampath *et al.*, 1995; Pencolé and Cordier, 2005; Ye and Dague, 2010; Lamperti and Zanella, 2011; Zhao *et al.*, 2012]. Candidate diagnoses have originally been defined as the trajectories resulting from the synchronization of the DES model with the given observation: faulty events can be extracted from such trajectories. The set of faults extracted from a trajectory is a diagnosis itself, at a higher abstraction level. Usually, *all* possible diagnoses, these being (possibly *non-minimal*) sets of faults, are produced as output by traditional methods. However, in diagnosis of static systems [Reiter, 1987; de Kleer and Williams, 1987], non-minimal diagnoses are not generated, in compliance with *Occam's Razor*. The same has recently been proposed by [Grastien *et al.*, 2011; 2012]: a hypothesis space is defined according to a preference criterion establishing the order by which hypothesis are generated and tested. Furthermore, a general conflict definition is proposed unifying the two different frameworks of static and DES diagnosis. Eventually, conflicts are computed on-line and exploited to prune the hypothesis space reducing the number of tests performed. In this paper we review the framework proposed by Grastien et al., particularly focusing on hypotheses defined as sequences of faulty events and we propose some heuristics to further reduce the number of tests performed.

*This work was supported in part by NSFC under Grant No. 61003101, and Zhejiang Provincial Natural Science Foundation under Grant No. Y1100191.

2 Definition of Diagnosis

2.1 Sequential Diagnosis of DESs

A DES to be diagnosed is modeled by an automaton, which is represented as a 5-tuple Mod = (**S**, Σ, **T**, S_0, S_f), where:

S: the set of states of the DES;

Σ: the set of events of the DES. It includes two disjoint sets: Σ_o, denoting all the observable events that are perceived by observers (e.g. sensors); and Σ_u, denoting all the other events that are not observed directly. Thus we have: $\Sigma = \Sigma_o \uplus \Sigma_u$ (here "$\uplus$" denotes the *union* operator between two disjoint sets). In addition, Σ_u can be partitioned into Σ_f, denoting all the faulty events to be inferred (we assume $\Sigma_f = \{f_1, f_2, \ldots, f_n\}$ in this paper), and Σ_{uf}, denoting all the other unobservable events that are not faulty events. Then we have: $\Sigma_u = \Sigma_f \uplus \Sigma_{uf}$.

T: the set of all possible state transitions of the DES, $\mathbf{T} \subseteq \{(s_i, e, s_j) \mid s_i, s_j \in \mathbf{S}, \text{and } e \in \Sigma\}$;

S_0: $S_0 \subseteq \mathbf{S}$, is the set of all possible initial states of the DES;

S_f: $S_f \subseteq \mathbf{S}$, is the set of all possible final states of the DES.

Notation $\|Mod\| \subseteq \Sigma^*$ is introduced to denote the behavior of the considered DES, which is the language recognized by automaton Mod: each word of such a language, called *trace*, is a (possibly empty) sequence of events in Σ.

The following two assumptions are made:

(1) $\|Mod\|$ does not include any cycle of unobservable events [Sampath *et al.*, 1995];

(2) Between each pair of faulty events in a trace, there is at least an observable event.

A *diagnosis problem* inherent to the considered DES is a pair (Mod, OBS), where $OBS \subseteq \Sigma_o^*$ is a finite set of sequences of observable events. Because of uncertainty, there may be more than one observation sequences in OBS, each of which may be the real sequence of observable events that have occurred in the system. Let $OBS_{ref} = OBS \cap Prj_{\Sigma_o}(\|Mod\|)$ denote the sequences of observable events in OBS that are consistent with the behavior of DES Mod, where $Prj_{\Sigma_o}(\|Mod\|)$ represents the projection of $\|Mod\|$

on Σ_o, i.e., all the sequences in Mod stripped of unobservable events.

In the following, we assume $OBS_{ref} = \{o_1, o_2, \cdots, o_m\}$ ($m \geq 1$), to represent the current refined observation, where each o_i denotes an observation sequence.

Based on the notation of a diagnosis problem, we define the corresponding diagnosis here below.

A (possibly empty) sequence $\delta \in \Sigma_f^*$ of faulty events is called a *sequential candidate diagnosis* of diagnosis problem (Mod, OBS), if the following condition holds:

$$\exists o \in OBS_{ref}, \exists \sigma \in \|Mod\|, \; such\ that: \\ Prj_{\Sigma_f}(\sigma) = \delta \ \wedge \ Prj_{\Sigma_o}(\sigma) = o,$$

i.e., there exists a trace σ that can be projected both on the sequence δ of faulty events and on a sequence of observable events $o \in OBS_{ref}$.

In the following we will use predicate $Diag(Mod, \delta, o)$ to denote that δ is a sequential candidate diagnosis for DES Mod that fulfills the above condition for a given $o \in OBS_{ref}$.

We define the diagnosis set Δ_i related to $o_i \in OBS_{ref}$, and the set of all diagnoses Δ related to OBS_{ref} as follows:

$$\Delta_i = \{\delta | Diag(Mod, \delta, o_i) \; holds\}; \\ \Delta = \{\Delta_1 \cdots \Delta_m\}, for\ OBS_{ref} = \{o_1 \cdots o_m\},$$

i.e., each Δ_i is the set of diagnoses for $o_i \in OBS_{ref}$.

2.2 Minimal Sequential Diagnosis of DESs

We will focus our attention on the sequential diagnosis with minimal length only. Three different definitions of minimal sequential diagnosis are given as follows, relying on the different representations of the so-called "*minimality*".

(1) *Successive minimal diagnosis* (Δ_{succ}):

Δ is called a *successive minimal diagnosis* set if:

$$\forall \delta \in \Delta_i (1 \leq i \leq m), \nexists\ \delta^{'} \in \Delta_i, such\ that\ \delta^{'} \prec_{succ} \delta,$$

where, the preference relationship "$\preceq_{succ}$" between two candidate diagnoses δ and $\delta^{'}$ is defined as follows:

$$\delta \preceq_{succ} \delta^{'} \; holds \; if: \; \delta^{'} = u\delta v, \; u, v \in \Sigma_f^*; \; and \\ \delta \prec_{succ} \delta^{'} \; holds \; \; if: \; \delta \preceq_{succ} \delta^{'} \; and \; \delta \neq \delta^{'}.$$

For example, let $\delta_1 = f_1 f_2$, $\delta_2 = f_1 f_2 f_3$, then $\delta_1 \prec_{succ} \delta_2$.

(2) *Consistent minimal diagnosis* (Δ_{cons}):

Δ is called a *consistent minimal diagnosis* set if:

$$\forall \delta \in \Delta_i (1 \leq i \leq m), \nexists\ \delta^{'} \in \Delta_i, \; such\ that\ \delta^{'} \prec_{cons} \delta,$$

where the preference relationship "$\preceq_{cons}$" between two candidate diagnoses δ and $\delta^{'}$ is defined as follows:

$$\delta \preceq_{cons} \delta^{'} \; \; holds \; \; if: \; \; \delta = s_1 s_2 \ldots s_l \; and \\ \delta^{'} = t_1 s_1 t_2 s_2 \ldots t_l s_l t_{l+1} (s_j \in \Sigma_f \cup \{\varepsilon\}, \\ 1 \leq j \leq l; \; and \; \; t_i \in \Sigma_f^* \; \; (1 \leq i \leq l+1)); \; \; and \\ \delta \prec_{cons} \delta^{'} \; \; holds \; \; if: \; \; \delta \preceq_{cons} \delta^{'} \; and \; \delta \neq \delta^{'}.$$

For example, let $\delta_1 = f_1 f_2$, $\delta_3 = f_1 f_3 f_2$, then $\delta_1 \prec_{cons} \delta_3$.

(3) *Set-contain minimal diagnosis* (Δ_{setc}):

Δ is called a *set-contain minimal diagnosis* set if:

$$\forall \delta \in \Delta_i (1 \leq i \leq m), \nexists\ \delta^{'} \in \Delta_i, \; such\ that\ \delta^{'} \prec_{setc} \delta,$$

where the preference relationship "$\preceq_{setc}$" between candidate diagnoses δ and $\delta^{'}$ is defined as follows:

$$\delta \preceq_{setc} \delta^{'} \; holds \; if: \; MultiSet(\delta) \overset{M}{\subseteq} MultiSet(\delta^{'}); \\ and \; \delta \prec_{setc} \delta^{'} \; \; holds \; \; if: \\ \delta \preceq_{setc} \delta^{'} \; and \; MultiSet(\delta) \neq MultiSet(\delta^{'}),$$

where $MultiSet(\delta) = \{s_1, s_2, \cdots, s_l\}$ for $\delta = s_1 s_2 \cdots s_l$, denotes a multi-set composed by all events in δ. Note: it is possible that $s_i = s_j$ ($1 \leq i, j \leq l$, $i \neq j$), i.e., there may be duplicates in $MultiSet(\delta)$. In addition, given two *MultiSets* MS_1 and MS_2, $MS_1 \overset{M}{\subseteq} MS_2$ means that each event e with p occurrences in MS_1 must have at least p occurrences in MS_2. Moreover, $MS_1 \overset{M}{\subset} MS_2$ means that $MS_1 \overset{M}{\subseteq} MS_2$ and $MS_1 \neq MS_2$.

For example, let $\delta_1 = f_1 f_2$, $\delta_4 = f_3 f_2 f_1$, then $\delta_1 \prec_{setc} \delta_4$.

The three definitions of minimal diagnosis above are linked by some relationships.

Proposition 1. *For each observation $o_i \in OBS_{ref}$:*

$$\Delta_{setc_i} \subseteq \Delta_{cons_i} \subseteq \Delta_{succ_i}.$$

In the following we will focus our attention just on minimal sequential candidates δ whose length has a given (upper) limit k. Thus the number of faults in each candidate cannot be larger than k.

2.3 Hypothesis Candidate Space

Given Σ_f, let Σ_f^i ($0 \leq i \leq n$) denote the set of all possible sequences of i (not necessarily distinct) faulty events. The space $H(k)$ of all the sequences of faulty events having a length not greater than k is:

$$H(k) = \Sigma_f^0 \uplus \Sigma_f^1 \uplus \cdots \uplus \Sigma_f^k.$$

Therefore, the total number of sequences in $H(k)$ is:

$$n^0 + n^1 + n^2 + \cdots + n^k.$$

Then, the complexity for deriving $H(k)$ is $O(n^k)$, where k is a small integer usually. Diagnosing a DES now becomes a question of performing a search in $H(k)$ to find all minimal sequential diagnoses with length at most k.

3 Pruning SE-tree for Minimal Diagnosis

In this section, first, the concept of SE-tree is proposed; then an algorithm combining a pruning SE-tree is presented for deriving all minimal diagnoses.

3.1 SE-tree

An SE-tree (Sequence Enumeration tree) is a tree for progressively deriving all sequences in $H(k)$, starting from the empty sequence ε.

Given a set $S = \{e_1, e_2, \cdots, e_{|S|}\}$ ($|S|$ denotes the length of S), an SE-tree on S is inductively defined as follows:

1) The root node of the tree is decorated with ε;

2) Each node $\mathcal{N}$ has $|S|$ child nodes. The decoration δ_{c_i} for the i^{th} ($1 \leq i \leq |S|$) child node $\mathcal{N}_{c\_i}$ of $\mathcal{N}$, is defined as: $\delta_{c\_i} = \delta \cdot e_i$, i.e., $\mathcal{N}_{c\_i}$ is decorated with the concatenation of its parent node decoration δ and the i^{th} element e_i in S.

An SE-tree relevant to a set S is endowed with some properties.

- In the 0^{th} level of the SE-tree, there is only 1 (n^0) node: the root node decorated with ε;

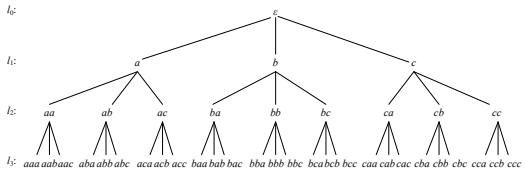


Figure 1: SE-tree with S= {a, b, c} and k = 3.

- In the i^{th} level of the SE-tree, there are n^i nodes, each of which is decorated with a sequence of i elements of S;
- All the possible sequences of elements of S with length i are in the i^{th} level of the tree.

Based on such properties, the total number of nodes of the SE-tree relevant to set S, with the length of each node decoration not greater than k ($k \geq 0$), is:

$$|S|^0 \ + \ |S|^1 \ + \ \ldots \ + \ |S|^k \ .$$

The complexity of creating such an SE-tree is $O(|S|^k)$.

Example 1. Given $S = \{a, b, c\}$, the SE-tree on S, where the maximum length of each node decoration is 3, is depicted in Fig. 1. ◇

3.2 Deriving All Minimal Diagnoses by SE-tree

Given a diagnosis problem (Mod, OBS), for each $o_i \in OBS_{ref}$, we derive all minimal sequential diagnoses with length k at most by *pruning* the relevant SE-tree over the set Σ_f with **Algorithm 1**.

At the beginning, **Step 1** is to create the root node (ε) of the SE-tree, i.e., we check whether the system is working normally or not if sequence o_i has been observed. If $Diag(Mod, \varepsilon, o_i)$ holds, i.e., a normal behavior of the system is consistent with o_i, the only faulty sequence will be ε in Δ_i. Then the algorithm has finished.

Otherwise, there will be at least one faulty event in the system. Then we start to extend the SE-tree from level 1 to level k. For each level, all possible fault sequences are generated (**Step 2**).

Then, we have to check whether each newly generated fault sequence δ_{c_r} is a minimal diagnosis of the current observation o_i (**Step 3**, by finding whether there is a minimal fault sequence with a shorter length than δ_{c_r}, or else by calling "$Diag(Mod, \delta_{c\_r}, o_i)$" to test).

In the end, all minimal diagnoses with length at most k will have been generated, each of which is the decoration of a node marked with "✓". Note that the preference relation "$\prec$" in **Step 3** can be any each one in set {$\prec_{succ}$, $\prec_{cons}$, $\prec_{setc}$} defined in Subsection 2.2.

All minimal sequential diagnoses with length at most k for OBS_{ref} will be obtained by running **Algorithm 1** for $|OBS_{ref}|$ times, one for each $o_i \in OBS_{ref}$.

Some analysis of **Algorithm 1** is sketched here below:

- *Correctness*: On the one hand, any sequence decorating a node marked with "✓" in the SE-tree, has been tested by $Diag$ to be *True* (**Step 1** or **Step 3**), so it is necessarily a diagnosis for o_i. On the other hand, in **Step 3**, any sequence decorating a node marked with "✓" in the SE-tree, must be a minimal diagnosis, because any non-minimal diagnosis has been marked with "×" to prevent any further extension.
- *Completeness*: In **Step 1**, ε in level 0 is generated and tested. In **Step 2**, for r ranging from 1 to n, all possible fault sequences with length that is less than or equal to k are generated in each level, if necessary, according to the definition of SE-tree. Thus, **Algorithm 1** is complete.
- *Complexity*: According to the properties of the SE-tree, the number of nodes generated by **Algorithm 1** is $O(n^k)$. However, the experimental complexity can be reduced considerably with respect to this upper bound by two pruning rules as follows:
  - If the sequence of faults δ, generated at level l_j ($1 \leq j \leq k$) of the SE-tree, is a minimal diagnosis (marked with "✓"), then all of its descendant nodes will neither be generated nor tested, and the number of pruned nodes is $\sum_{i=1}^{k-j} n^i$.
  - All nodes whose decoration δ' is not preferable to a minimal diagnosis δ ($\delta \prec \delta'$), will not be tested by calling $Diag$, instead they will be marked by "×". Thus their descendant nodes will not be generated. For each such δ' in level l_s ($1 \leq s \leq k$), the number of pruned nodes is $\sum_{j=1}^{k-s} n^j$.

Algorithm 1 (Deriving all minimal sequential diagnoses with length at most k for o_i by SE-tree)

Input: A DES model Mod with faulty events Σ_f = {f_1, f_2, $\cdots$, f_n}, an observation sequence $o_i \in OBS_{ref}$;

Step 1: In level l_0, create the root node decorated with ε;
  Init Δ_i = { };
  If $Diag(Mod, \varepsilon, o_i)$ holds, **Then**
    $\Delta_i = \Delta_i \cup \{\varepsilon\}$;
    Mark the root node with "✓";
    Return Δ_i;
  EndIf //End for the root node
  For level index j = 1 to k //Extending the SE-tree
Step 2://Generating all sequences in level l_j($1 \leq j \leq k$):
    For each node $\mathcal{N}$ (suppose it is decorated with δ) in level $l_{(j-1)}$ **not** marked as "×" or "✓"
      For r = 1 to n
        Generate $\mathcal{N}$'s r^{th} child node $\mathcal{N}_{c\_r}$, and decorate it with δ_{c_r}: $\delta_{c\_r} = \delta \cdot f_r$;
      EndFor //End of generating all children of $\mathcal{N}$
    EndFor
Step 3: //Applying pruning rules to each node //generated at Step 2:
3.01 **For each** node $\mathcal{N}_{c\_r}$ decorated with faults δ_{c_r}
3.02 $flag$ = 0;
3.03 **For each** node $\delta_s \in \Delta_i$ with length less than j
3.04 **If** $\exists\ \delta_s \in \Delta_i$, such that: $\delta_s \prec \delta_{c\_r}$, **Then**
3.05 Mark $\mathcal{N}_{c\_r}$ with "×" to stop its extension;
3.06 $flag$ = 1;
3.07 **EndIf**
3.08 **EndFor**
3.09 **If** $flag$ == 0, i.e., $\nexists \delta_s \in \Delta_i$ such that: $\delta_s \prec \delta_{c\_r}$,
3.10 **Then**
3.11 **If** $Diag(Mod, \delta_{c\_r}, o_i)$ holds, **Then**
3.12 $\Delta_i = \Delta_i \cup \{\delta_{c\_r}\}$;
3.13 Mark $\mathcal{N}_{c\_r}$ with "✓"; (stop its extension)
3.14 **EndIf**
3.15 **EndIf**
3.16 **EndFor** //End of checking via pruning rules
  EndFor //End of extending the SE-tree till level k

Output: Δ_i: The minimal sequential diagnoses for o_i.

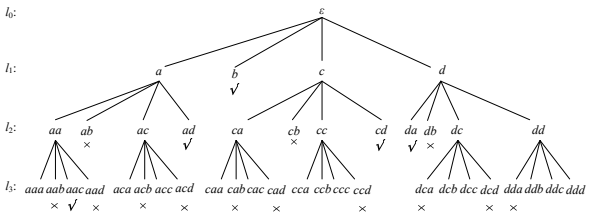


Figure 2: SE-tree for diagnoses b, ad, cd, da, and aac.

Example 2. Suppose Σ_f = {a, b, c, d} in a DES with model Mod, and that all minimal sequential diagnoses whose length is k = 3 at most are b, ad, cd, da, and aac, for an observation sequence o. Then, the SE-tree for deriving all such minimal sequential diagnoses is shown in Fig. 2, where all the subtrees whose roots are the nodes marked by "×" or "✓" have been pruned. ◇

3.3 Diag(Mod, δ, o)

Given the model Mod of a DES, and an observation sequence o, predicate $Diag$ decides whether the fault sequence δ is a candidate diagnosis or not. In order to implement the predicate, usually a diagnosis problem (Mod, $\{o\}$) is transformed into a corresponding intelligent planning problem or a proposition satisfiability (SAT) problem. Hence, the output of predicate $Diag$ will be obtained by using state-of-the-art planners or SAT-solvers. (For more details about the transformation, please refer to [McIlraith, 1994; Sohrabi *et al.*, 2010; Grastien *et al.*, 2011; 2012] etc).

The efficiency of the process for deriving all minimal sequential diagnoses is considerably affected by the efficiency of the currently used solvers.

4 Heuristic Information

In this section, some heuristic information is investigated for generating fewer nodes of an SE-tree, so as to improve the efficiency.

4.1 Extended Diagnosability of DESs

Diagnosability analysis of a DES is mainly used to decide whether some faults have definitely occurred or not [Sampath *et al.*, 1995]. In the following, an extended definition of diagnosability of a DES is given.

Definition 1. *(Extended diagnosability) A DES with model Mod is extendedly diagnosable, iff $\nexists\ \sigma_1, \sigma_2 \in \|Mod\|$ with $\sigma_1 \neq \sigma_2$, such that:*

$$Prj_{\Sigma_o}(\sigma_1) = Prj_{\Sigma_o}(\sigma_2)$$
$$and\ Prj_{\Sigma_f}(\sigma_1) \neq Prj_{\Sigma_f}(\sigma_2).$$

In other words, if a DES is extendedly diagnosable, there will not exist any pair of trajectories that have the same observation sequence and different fault sequences.

Proposition 2. *Given a DES with model Mod, if it is extendedly diagnosable, then for each observation sequence $o \in Prj_{\Sigma_o}(\|Mod\|)$, there is one and only one minimal diagnosis $\delta \in \Sigma_f^*$ for o.*

Based on Proposition 2, if the DES is extendedly diagnosable, once we have found a fault sequence for an observation sequence o, we can just stop searching (or testing) any other fault sequence.

Although it is sometimes difficult to check the extended diagnosability of the whole DES, it may be easy to check whether some fault sequences are extendedly diagnosable or not. Therefore, we give a definition of (extended) diagnosability of a fault sequence as follows.

Definition 2. *(Diagnosability of a fault sequence) Given a DES with model Mod, a fault sequence $\delta \in \Sigma_f^*$ is diagnosable, iff $\nexists\ \sigma_1, \sigma_2 \in \|Mod\|$ with $\sigma_1 \neq \sigma_2$, such that:*

$$Prj_{\Sigma_o}(\sigma_1) = Prj_{\Sigma_o}(\sigma_2) = o$$
$$and\ \ Prj_{\Sigma_f}(\sigma_1) = \delta,\ Prj_{\Sigma_f}(\sigma_2) \neq \delta.$$

Based on the notion of diagnosability of a fault sequence, we can write a proposition similar to the previous one as follows.

Proposition 3. *Given a DES with model Mod, if a fault sequence $\delta \in \Sigma_f^*$ is diagnosable, then for each observation sequence $o \in Prj_{\Sigma_o}(\|Mod\|)$, if δ is a minimal diagnosis for o, then δ is the only minimal diagnosis for o.*

Thus, if a fault sequence δ is diagnosable, once we have found out that δ is a diagnosis for o, we will immediately stop the extension of the SE-tree for observation o.

4.2 Conflict Set

Definition 3. *(Conflict set) Given a DES with model Mod, a set CS = {$\neg f_1, \neg f_2, \cdots, \neg f_t$} is called a conflict set for a nonempty observation sequence $o \in \Sigma_o^+$, iff for all sequential diagnoses $\delta_1, \cdots, \delta_s \in \Sigma_f^*$ for o, at least one fault f_j ($1 \leq j \leq t$) is in each δ_i ($1 \leq i \leq s$).*

Remark. The concept of "conflict set" introduced above is inspired by the notion of a conflict in diagnosis of static systems [Reiter, 1987], this being a set of components, such that it is *impossible* that all of them are working normally. Predicate $AB(x)$ denotes that component x is working *ab*normally, then $AB(x)$ is just like a faulty event of a DES. Set $\{c_1, c_2, \cdots, c_l\}$ is called a conflict if, $\{\neg AB(c_1), \neg AB(c_2), \cdots, \neg AB(c_l)\}$ is *inconsistent* with the current observation.

Our definition of a "conflict set" can be seen as a specialization of the traditional notion of a conflict: given the current observation sequence o, it is *impossible* that *no* fault in {$f_1, f_2, \cdots, f_t$} has occurred.

If {$\neg f_1, \neg f_2, \cdots, \neg f_t$} is a conflict set for observation o, then at least one fault in {$f_1, f_2, \cdots, f_t$} will be included in a candidate diagnosis. If the decoration of a node does not include any fault belonging to the set {$f_1, f_2, \cdots, f_t$} relevant to observation sequence o, then such a decoration cannot be a diagnosis consistent with o, thus it is useless calling a $Diag$ solver to check. Therefore, the number of calls to $Diag$ solver will be reduced. Moreover, the conflict set is also a heuristic rule for pruning an SE-tree.

In practice, for each observation sequence o, if we know that some f_i are necessarily faults for o in some situations, then all the $\neg f_i$ will be composed as a conflict set for o.

4.3 Impossible Fault Sequence

Definition 4. *(Impossible fault sequence, IFS) A sequence $\delta_{IFS} \in \Sigma_f^*$ is an impossible fault sequence, iff $\delta_{IFS} \notin Prj_{\Sigma_f}(\|Mod\|)$.*

An IFS δ_{IFS} is just a fault sequence inconsistent with the system model.

Proposition 4. *Given a DES with model Mod, suppose the minimal sequential diagnosis set is Δ = {$\Delta_1 \cdots \Delta_m$} for refined observations OBS_{ref} = {$o_1 \cdots o_m$}. For any IFS δ_{IFS}, we have $\delta_{IFS} \notin \Delta_i$ for any $i \in$ {$1 \cdots m$}.*

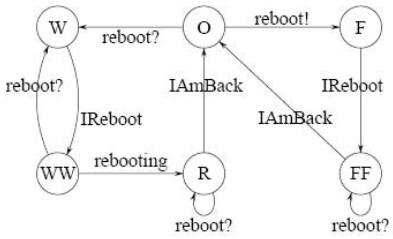
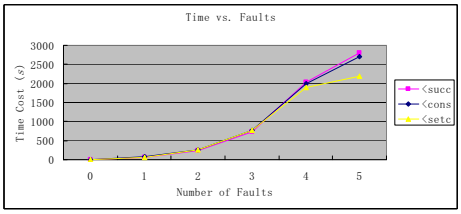
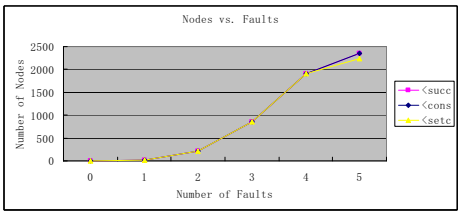


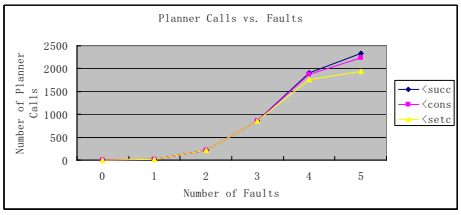
Figure 3: Component model.



(a) Average time vs. length of fault sequence.



(b) Average nodes vs. length of fault sequence.



(c) Average planner-calls vs. length of fault sequence.

Figure 4: Experimental results.

Proposition 5. *Given a DES with model Mod, if a fault sequence $\delta \in Prj_{\Sigma_f}(\|Mod\|)$, then any prefix of δ is in $Prj_{\Sigma_f}(\|Mod\|)$.*

Corollary 1. *$Prj_{\Sigma_f}(\|Mod\|)$ does not include any fault sequence with an IFS prefix.*

Therefore, when extending an SE-tree, if a node decorated with an IFS is generated, then the node can be marked with "×" immediately, without comparing it with previously found minimal diagnoses, or testing it by the $Diag$ solver at all. IFSs can be generated off-line, as heuristic rules for improving the efficiency of on-line diagnosis.

5 Experiments

In order to prune the hypothesis space and reduce the number of calls to the test solver, the implemented approach has been combined with a *Look-ahead (LA)* technique. Such a technique is inspired by [Grastien *et al.*, 2012] but, differently from the quoted paper, it is based on planning technology and exploits a suitable system model encoding to represent hypotheses, instead of representing them symbolically. Given a hypothesis h, the planner looks for a trace $\sigma \in \|Mod\|$ where $\delta(\sigma) = h'$, $h \preceq h'$, and minimizes the *distance* between h and h' defined as the difference between their depths. If no such a trace can be found (and the used planner is complete), then neither h nor any of its descendants is a candidate, thus the sub-tree rooted in h can be pruned. Otherwise, if a trace is found and its metric value is zero, then h is a minimal candidate and all its descendants are pruned; or else, if a found plan has a metric value mv greater than zero (and the used planner is optimal), the current hypothesis h need to be expanded. In other words, h is the prefix of an unknown minimal candidate and a further descent in the hypothesis space is needed. In a word, we use a planner to implement the predict $Diag$ (and then $Diag(Mod, \delta, o)$ becomes a function with 3 kinds of return values: δ, $\emptyset$, and a proper prefix of δ). Specifically, the codes 3.11 - 3.14 in Algorithm 1 are replaced with the following codes in practice.

If $Diag(Mod, \delta_{c\_r}, o_i) = \delta_{c\_r}$, **Then**
  $\Delta_i = \Delta_i \cup \{\delta_{c\_r}\}$;
  Mark $\mathcal{N}_{c\_r}$ with "✓"; (stop its extension)
Else If $Diag(Mod, \delta_{c\_r}, o_i) = \emptyset$, **Then** //i.e., $mv = 0$
  Mark $\mathcal{N}_{c\_r}$ with "×"; (stop its extension)
EndIf

Also, "$Diag(Mod, \varepsilon, o_i)$ holds" is replaced with "$Diag(Mod, \varepsilon, o_i) = \varepsilon$" in **Step 1** of Algorithm 1.

We use the DES D, introduced by [Grastien *et al.*, 2007] and adopted also in [Sohrabi *et al.*, 2010], which consists of 20 identical components (e.g. computers) in a 5×4 grid. Each component has four neighbors; corner and border components are neighbors to components on their opposite sides. Fig. 3 provides an automaton representation of the behavior of one component. The initial state of each component is O. When it fails, its state changes from O to F and sends a message $reboot!$ to its four neighbors. The neighbors receive the message $reboot?$ and change their state from O to W, from WW to W, from R to R or from FF to FF, depending on their current state. Note that there is (only) a faulty event per component, thus the set of faulty events Σ_f globally includes 20 events. Events $IReboot$ and $IAmBack$ are the only observable ones. The sender component of each observed event is assumed to be known, thus Σ_o globally includes 20 * 2 = 40 events.

The observation considered in each test case includes 10 observable events, each of which has been randomly chosen from alphabet Σ_o. The temporal uncertainty degree of an observation that includes 10 observable events may vary from 0 (the 10 observed events are totally temporally ordered by 45 precedences) to 45 (all the events are temporally unrelated, since all the precedences have been removed). A (physically possible 5) observation for each uncertainty degree was generated, thus obtaining 46 different observations O and, therefore, 46 distinct test cases, each being a diagnosis problem (D, O).

We set k = 5, i.e., the maximal length of each fault sequence is 5. We tested all three preference relations $\prec_{succ}$, $\prec_{cons}$, and $\prec_{setc}$. The complete planner run in our experiments is MFF [Hoffmann, 2003], whose input is given in PDDL 2.1 level 2. Each observation is compiled into the planning problem in the way presented in [Grastien *et al.*, 2007; Sohrabi *et al.*, 2010]. The exploited hardware platform is a 64-bit PC, endowed with a single core AMD NEO 2,2 GHz microprocessor and a 4 GB RAM. The *OS* is Linux Ubuntu 12.04 LTS and the platform is Java 7 SE.

The experimental results are shown in Fig. 4. From Fig. 4, we can see that all three kinds of minimal sequence diagnoses are derived quickly when the length is small. In

addition, the time cost is considerably affected by the number of nodes and the number of calls to the planner. Pruning rules described in Section 4 are not been implemented due to lack of time, but they can further reduce the number of calls to the solver.

6 Related Work

The computation of minimal diagnoses through the generation and exploration of an hypothesis space has recently been proposed by Grastien *et al* [Grastien *et al.*, 2011; 2012]. These papers exploiting a SAT solver to determine candidates while our reimplementation uses a planner. Again, our approach explicitly defines a simple SE-tree and exploits pruning rules to prevent the generation of unconsistent hypothesis. In addition, we state a specialized notion of "conflict set" as a heuristic rule to accelerate diagnosis in the context of the SE-tree. Furthermore, an extended diagnosability for a fault sequence is proposed for pruning an SE-tree, while traditional diagnosability of DESs is only for a faulty event [Sampath *et al.*, 1995].

The approach is different from the history-based approach [Lamperti and Zanella, 2003], which firstly synchronizes the observations with the DES model, and then extracts all diagnoses, including non-minimal diagnoses. Our approach instead tries to synchronize the DES model with minimal candidate fault sequences only, and to verify whether each of such sequences is a diagnosis or not.

Our approach is also different from the conflict-directed A^* algorithm for deriving the most likely static candidates [Williams and Ragno, 2007]. They rank hypotheses as we do, but they exploit probability information to guide the search towards most probable diagnosis. We don't use probability and we propose to rank hypotheses on the basis of several preference criteria.

7 Conclusion

A framework for model-based diagnosis of DESs is proposed, which is focused on minimal sequential diagnosis. The concept of SE-tree is put forward, and all minimal diagnoses can be derived efficiently by combining a pruning SE-tree. The concepts of extended diagnosability, conflict set in DESs, and impossible fault sequences are presented to be exploited as the basis for heuristic rules to improve the efficiency of diagnosis computation.

More heuristics defined in Section 4 are needed to be evaluated to reduce calling a planner in the future (note: this paper mainly concerns the theoretical soundness of the heuristics). Further aspects to investigate also involve the decomposition of a problem in several sub-problems and then recompose the local solutions found in global ones on the basis of heuristic information generated off-line.

Remark *All propositions and corollaries can be formally proven; proofs have been omitted for space reasons.*

Acknowledgment

We owe a debt of gratitude to anonymous reviewers for very useful comments on the paper.

References

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[Grastien *et al.*, 2007] Al. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, pages 305–310, Vancouver, British Columbia, Canada, 2007.

[Grastien *et al.*, 2011] Al. Grastien, P. Haslum, and S. Thiébaux. Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX-11)*, pages 60–67, Murnau, Germany, 2011.

[Grastien *et al.*, 2012] Al. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: theory and practice. In *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR-12)*, pages 489–499, Rome, Italy, 2012.

[Hoffmann, 2003] Jörg Hoffmann. The metric-ff planning system: translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of Active Systems: Principles and Techniques*. Springer-Verlag New York, Inc., USA, 2003.

[Lamperti and Zanella, 2011] G. Lamperti and M. Zanella. Context-sensitive diagnosis of discrete-event systems. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 969–975, Barcelona, Spain, 2011.

[McIlraith, 1994] S. McIlraith. Generating tests using abduction. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 449–460, Bonn, Germany, 1994.

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

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

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40:1555–1575, 1995.

[Sohrabi *et al.*, 2010] S. Sohrabi, J. Baier, and S. McIlraith. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR-10)*, pages 26–36, Toronto, Canada, 2010.

[Williams and Ragno, 2007] B. C. Williams and R. J. Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathmatics*, 155:1562–1595, 2007.

[Ye and Dague, 2010] L. Ye and P. Dague. Diagnosability analysis of discrete event systems with autonomous components. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, pages 105–110, Lisbon, Portugal, 2010.

[Zhao *et al.*, 2012] X. Zhao, D. Ouyang, L. Zhang, X. Wang, and Y. Mo. Reasoning on partially-ordered observations in online diagnosis of dess. *AI Communications*, 25:285–294, 2012.

Hybrid automaton incremental construction for online diagnosis *

Jorge Vento[1], **Louise Travé-Massuyès**[2,3], **Ramon Sarrate**[1] and **Vicenç Puig**[1]
[1]Advanced Control Systems (SAC), Universitat Politècnica de Catalunya (UPC), Barcelona,Spain
e-mail: {jorge.isaac.vento,ramon.sarrate,vicenc.puig}@upc.edu
[2] CNRS, LAAS, 7, avenue du Colonel Roche, F-31400 Toulouse, France
[3] University of Toulouse, LAAS, F-31400 Toulouse, France
e-mail: louise@laas.fr

Abstract

This paper proposes a method to track the system mode and diagnose a hybrid system without building an entire diagnoser off-line. The method is supported by a hybrid automaton model that represents the hybrid system continuous and discrete behavioral dynamics. Diagnosis is performed by interpreting the events and measurements issued by the physical system directly on the hybrid automaton model. This interpretation leads to building the useful parts of the diagnoser incrementally, developing only the branches that are required to explain the occurrence of the incoming events. The resulting diagnoser adapts to the system operational life and is much less demanding in terms of memory storage. The proposed framework subsumes previous works in that it copes with both structural and non-structural faults. The method is validated on an application case study based on the sewer network of the Barcelona city.

1 Introduction

The approaches to detect and isolate faults in hybrid systems have been addressed by both the FDI and DX communities. In the FDI approach, the diagnosis is based on the hybrid automaton to track the system mode [Bayoudh *et al.*, 2008; Vento *et al.*, 2011] combining the continuous and discrete techniques to detect and isolate faults. On the other hand, in the DX approach some authors have proposed alternative ways to diagnose hybrid systems like using the hybrid bond graph formalism [Narasimhan and Biswas, 2007; Daigle, 2008].

This paper follows the work presented in [Vento *et al.*, 2011; Bayoudh *et al.*, 2008] where parity-space residuals are used to track the mode and diagnose the hybrid system. In [Vento *et al.*, 2011], the operation modes represent nominal behavior and diagnosis focuses on fault detection and isolation of *non-structural faults*, i.e. faults that do not change the structure of the model (e.g. additive faults in sensors and actuators). In [Bayoudh *et al.*, 2008], the operation modes may be nominal or faulty, leading to the capability of detecting and isolating *structural faults* (e.g. an actuator stuck at a given position, opened or closed). In both cases, a set of analytical redundancy relations (ARR) are inferred from the set of equations in each mode and they are used to generate a set of residuals. In the case of non-structural faults, the fault effect on the residuals of every mode is assumed to be known and is captured by theoretical fault signatures. Tracking the system mode involves detecting that the residuals of the current mode are different from zero and checking the theoretical fault signatures against the residuals evaluated with measurements. In the case of structural faults, fault models are assumed to be known and the residuals of a faulty mode are expected to become zero when the fault is present.

The methods presented in the above mentioned works rely on a finite state machine called a *diagnoser* [Sampath *et al.*, 1995] which is built off-line from the hybrid model and the residuals are generated for each mode as explained in [Vento *et al.*, 2011; Bayoudh *et al.*, 2008]. The main issue with these off-line approaches is that since the number of states of the diagnoser grows exponentially with the number of states of the hybrid automaton, the generation of the set of residuals for every mode may be a limiting factor.

This paper proposes a method to track the system mode and diagnose the hybrid system without building the entire diagnoser off-line. Diagnosis is performed by interpreting the events and measurements issued by the physical system directly on the hybrid automaton model. This interpretation leads to building the useful parts of the diagnoser incrementally, developing only the branches that are required to explain the occurrence of the incoming events. Generally, a hybrid system operates in a small region compared to the entire behavioral space defined by the hybrid automaton states. A significant gain hence comes from the proposed approach. Moreover, the proposed framework subsumes previous works in the sense that structural and non-structural faults are considered at the same time.

The structure of the paper is the following. In Section 2, the hybrid model is presented. Section 3 provides the principles of the proposed method to diagnose faults in hybrid systems. In Section 4, the method to incrementally build the diagnoser of the hybrid system is presented as well as its implementation. In Section 5, an application case study based on the sewer network of the Barcelona city is used to assess the validity of the proposed approach. Finally, conclusions are given in Section 6.

*This work has been funded by the Spanish Ministry of Science and Technology through the CICYT project WATMAN (ref. DPI2009-13744) and by the Spanish Ministry of Economy and Competitiveness through the CICYT project SHERECS (ref. DPI2011-26243)

2 Hybrid System Modeling

The hybrid automaton model results from an adaptation of [Lygeros *et al.*, 2003; Bayoudh *et al.*, 2008; Vento *et al.*, 2011]. This work assumes linear continuous dynamics in each mode represented by discrete-time state space models. Let us consider that the model of the hybrid system to be diagnosed can be described by the following hybrid automaton $HA =< \mathcal{Q}, \mathcal{X}, \mathcal{U}, \mathcal{Y}, \mathcal{F}, \mathcal{G}, \mathcal{H}, \Sigma, \mathcal{T} >$, where:

- $\mathcal{Q}$ is a set of modes. Each $q_i \in \mathcal{Q}$ with $|\mathcal{Q}| = n_q$ represents a nominal operation or faulty mode [1] of the system such that $\mathcal{Q} = \mathcal{Q}_{\mathcal{N}} \cup \mathcal{Q}_{\mathcal{F}_s}$.
- $q_0 \in \mathcal{Q}$ is the initial mode.
- $\mathcal{X} \subseteq \mathcal{R}^{n_x}$ defines the continuous state space. $\mathbf{x}(k) \in \mathcal{X}$ is the discrete-time state vector and $\mathbf{x}_0$ the initial state vector.
- $\mathcal{U} \subseteq \mathcal{R}^{n_u}$ defines the continuous input space. $\mathbf{u}(k) \in \mathcal{U}$ is the discrete-time input vector.
- $\mathcal{Y} \subseteq \mathcal{R}^{n_y}$ defines the continuous output space. $\mathbf{y}(k) \in \mathcal{Y}$ is the discrete-time output vector.
- $\mathcal{F}$ is the set of faults that can be partitioned as $\mathcal{F} = \mathcal{F}_s \cup \mathcal{F}_{ns}$ taking into account structural[2] and non-structural faults.
- $\mathcal{G}$ defines a set of discrete-time state affine functions for each mode:

$$\mathbf{x}(k+1) = \mathbf{A}_i\mathbf{x}(k) + \mathbf{B}_i\mathbf{u}(k) + \mathbf{F}_{xi}\mathbf{f}_{ns}(k) + \mathbf{E}_{xi} \quad (1)$$

  where $\mathbf{A}_i \in \mathcal{R}^{n_x \times n_x}$, $\mathbf{B}_i \in \mathcal{R}^{n_x \times n_u}$ and $\mathbf{E}_{xi} \in \mathcal{R}^{n_x \times 1}$ are the state matrices in mode q_i, $\mathbf{f}_{ns}(k)$ is a vector representing non-structural faults with $\mathbf{F}_{xi}$ being the fault distribution matrix. The case $\mathbf{f}_{ns}(k) = 0$ corresponds to a nominal behaviour.
- $\mathcal{H}$ defines a set of discrete-time output affine functions for each mode:

$$\mathbf{y}(k) = \mathbf{C}_i\mathbf{x}(k) + \mathbf{D}_i\mathbf{u}(k) + \mathbf{F}_{y_i}\mathbf{f}_{ns}(k) + \mathbf{E}_{y_i} \quad (2)$$

  where $\mathbf{C}_i \in \mathcal{R}^{n_y \times n_x}$, $\mathbf{D}_i \in \mathcal{R}^{n_y \times n_u}$ and $\mathbf{E}_{y_i} \in \mathcal{R}^{n_y \times 1}$ are the output matrices in mode q_i and $\mathbf{F}_{y_i}$ is the fault distribution matrix.
- $\Sigma = \Sigma_s \cup \Sigma_c \cup \Sigma_{\mathcal{F}_s}$ is the set of events. Spontaneous mode switching events (Σ_s), input events (Σ_c) and structural fault events ($\Sigma_{\mathcal{F}_s}$) are considered. Σ can be partitioned as $\Sigma_o \cup \Sigma_{uo}$ where Σ_o represents the set of observable events and Σ_{uo} represents the set of unobservable events. $\Sigma_{\mathcal{F}_s} \subseteq \Sigma_{uo}$, $\Sigma_c \subseteq \Sigma_o$ and Σ_s may have elements in both sets.
- $\mathcal{T} : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is the transition function. The transition from mode q_i to mode q_j labeled with an event $\sigma \in \Sigma$ is denoted by $\mathcal{T}(q_i, \sigma) = q_j$ or by t_{ij} when the event is of no interest [3].

Alternatively, the model given by (1)-(2) can be expressed in input-output form using the delay operator which is denoted by p^{-1} and considering zero initial conditions as follows:

$$\mathbf{y}(k) = \mathbf{M}_i(p^{-1})\mathbf{u}(k) + \mathbf{\Upsilon}_i(q^{-1})\mathbf{f}_{ns}(k) + \mathbf{E}_{mi}(p^{-1}) \quad (3)$$

where p^{-1} is the delay operator, $\mathbf{M}_i(p^{-1})$ represents the transfer function between inputs and outputs of the system, $\mathbf{\Upsilon}_i(p^{-1})$ is the non-structural fault transfer function and $\mathbf{E}_{mi}(p^{-1})$ is a constant term.

[1] Faulty modes represent structural fault models.

[2] Every faulty mode $q_i \in Q_{\mathcal{F}_s}$ has a corresponding structural fault $f_{q_i} \in \mathcal{F}_s$ and is associated with a fault event defined in the set Σ_f.

[3] It is assumed that there is only one transition from a given mode q_i to a given mode q_j.

3 Proposed Hybrid Diagnosis Method

3.1 Principles of the Method

Model-based diagnosis is based on the use of a model of the monitored dynamic system to detect and isolate faults. The estimated system behaviour obtained from the system model is compared with the real behaviour available through sensor measurements. In particular, FDI algorithms for hybrid systems take into account which is the current operation mode to generate the set of residuals, used to build consistency indicators, and to achieve the diagnosis task [Vento *et al.*, 2011]. The scheme of the proposed method to diagnose hybrid systems is shown in Fig. 1. The figure shows the different tasks involved in online diagnosis. The original idea is to build an hybrid diagnoser in an incremental manner when an event occurs.

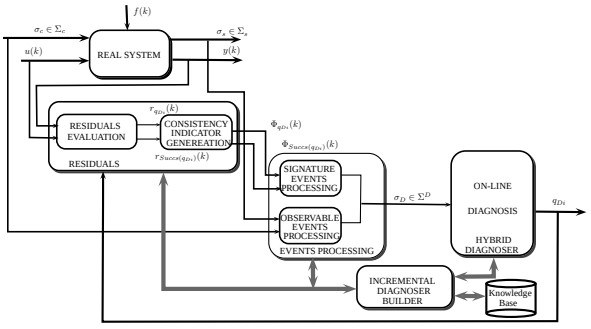


Figure 1: Conceptual block diagram for the proposed methodology

The method consists in tracking the mode sequence synchronously thanks to a diagnoser which is built incrementally from the so-called behaviour automaton, considering the possible current modes of the system and their successors. The original idea of the paper is to build the diagnoser in an incremental manner when an event occurs. The behaviour automaton includes so called signature events that abstract the residual behaviors. The transitions labelled by unobservable events in HA may hence turn observable by means of the signature events thanks to the discernability property (cf. section III.B).

To detect and isolate faults in the system two possibilities exist. On one hand, structural faults which produce changes in the dynamics are included in HA as faulty modes with their own dynamic model. Therefore, the corresponding mode is recognized when its consistency indicators are in agreement with measurements. On the other hand, non-structural faults are represented as disturbances on the models of the different modes of HA. Using the fault sensitivity of these models, a fault signature matrix can be generated. Then, a consistency test using this matrix is carried out comparing the set of observed consistency indicators with the columns of the fault signature matrix.

Diagnosis is based on the single fault assumption during the detection phase. However, two faults can occur sequentially, as long as the first one corresponds to a structural fault and the second one to a non-structural fault. Moreover, it is assumed that there is a minimal time between state transitions according to the dwell time of HA.

3.2 Diagnosis Based on Residual Consistency Indicators

Diagnosis based on continuous dynamics relies on residual properties. The set of residuals for the mode q_i is given by:

$$\mathbf{r}_i(k) = \mathbf{y}(k) - \mathbf{G}_i(p^{-1})\mathbf{u}(k) - \mathbf{H}_i(p^{-1})\mathbf{y}(k) - \mathbf{E}_i(p^{-1}) \quad (4)$$

where $\mathbf{G}_i(p^{-1})$, $\mathbf{H}_i(p^{-1})$ and $\mathbf{E}_i(p^{-1})$ can be calculated for instance, using the parity space or observer approach (for more details see f.e. [Ding *et al.*, 2008])

Once the residuals have been generated, they are evaluated with the measurements against a threshold, providing one consistency indicator of the following form for each residual:

$$\varphi_i^l(k) = \begin{cases} 0 & \text{if } |r_i^l(k)| \leq \tau_i^l \\ 1 & \text{if } |r_i^l(k)| > \tau_i^l \end{cases} \quad (5)$$

where $l \in \{1, \cdots, n_{r_i}\}$, n_{r_i} is the number of residuals for mode q_i and τ_i^l is the threshold associated with the residual $r_i^l(k)$. The consistency indicators are then gathered in the vector $\Phi_{q_i}(k) = [\varphi_i^1(k), \cdots, \varphi_i^{n_{r_i}}(k)]$. Sumarizing, consistency indicator vector $\Phi_{q_i}(k)$ is built from the binarised residuals (5) of mode q_i evaluated with the measurements corresponding to the current mode of the system at time k.

An important property to track the system mode is discernability. Discernability between two modes is the property that assesses whether two modes can be distinguished based on continuous measurements. If two modes q_i and q_j are discernible and the system changes from mode q_i to mode q_j or viceversa, the sequence of signals $(\mathbf{u}(k), \mathbf{y}(k))$ change from being consistent with mode q_i to being consistent with mode q_j or viceversa. This property can be verified using the consistency indicators defined above [Mezyani, 2007]: two modes are discernible iff the set of consistency indicators satisfy $\Phi_{q_i}(k) \neq \mathbf{0}$ and $\Phi_{q_j}(k) = \mathbf{0}$ with measurements corresponding to mode q_j and viceversa.

If neither the consistency indicators of mode q_i nor those of mode q_j are in agreement with measurements it is assumed that a non-structural fault is affecting the system. This kind of faults are identified using the concept of fault sensitivity [Vento *et al.*, 2011], which is determined by the expression:

$$\mathbf{\Lambda}_i(p^{-1}) \quad = \quad (\mathbf{I} - \mathbf{H}_i(p^{-1}))\mathbf{\Upsilon}_i(p^{-1})$$

where $\mathbf{\Upsilon}_i(p^{-1})$ represents the non-structural fault transfer function between the input and the non-structural fault vector in (1)-(2).

In particular, given the fault sensitivity of the j^{th} residual with respect to the l^{th} fault denoted as $\Lambda_i(j,l)$ (i.e, the element (j,l) of the sensitivity matrix $\mathbf{\Lambda}_i$), the element (j,l) of the fault signature matrix is determined as follows:

$$\mathbf{FS}_i(j,l) = \begin{cases} 1 & \text{if } \mathbf{\Lambda}_i(j,l)(p^{-1}) \neq 0 \\ 0 & \text{if } \mathbf{\Lambda}_i(j,l)(p^{-1}) = 0 \end{cases} \quad (6)$$

i.e., if the j^{th} residual in mode q_i depends on the l^{th} fault, it is coded as a 1 and it is coded as a 0 otherwise. A non-structural faulty situation is detectable, i.e. discernible from the nominal mode, is its signature is different from 0. Two faulty situations corresponding to the l^{th} and the $l^{'th}$ non-structural faults are discernible iff their signatures are different, i.e. $\mathbf{FS}_i(\bullet, l) \neq \mathbf{FS}_i(\bullet, l^{'})$. Since in a hybrid system, residuals change with the mode, the fault sensitivity as well as the theoretical fault signature matrix depend on the mode.

If the situation is such that neither a structural fault nor a non-structural fault can be isolated, the system mode is assumed unknown. The diagnosis based on consistency indicators assumes that the residual dynamics have time to establish between two consecutive transitions.

4 Hybrid Diagnosis

The diagnoser of the hybrid system is a finite state machine built from the behavior automaton and used, on one hand, to perform on-line diagnosis and on the other hand, to check the diagnosability of the hybrid system as presented in [Bayoudh *et al.*, 2008]. The method proposes to incrementally build the hybrid diagnoser from the behaviour automaton obtained while the system is monitored.

4.1 Behaviour Automaton

The behaviour automaton is the finite state generator of the abstract language $L(HA)$ resulting from abstracting the continuous dynamics captured by the residual consistency indicators in terms of discrete signature-events [Bayoudh *et al.*, 2008]. The behaviour automaton is defined by $B =< \overline{\mathcal{Q}}, \overline{\Sigma}, \overline{\mathcal{T}}, \overline{q}_0 >$.

- $\overline{\mathcal{Q}} = \mathcal{Q} \cup \mathcal{Q}_t \cup \mathcal{Q}^t_{\mathcal{F}_{ns}} \cup \mathcal{Q}_{\mathcal{F}_{ns}}$ is the set of discrete states where:
  - $\mathcal{Q}$ is the set of system modes,
  - $\mathcal{Q}^t$ is the set of transient modes between two discernible modes in HA,
  - $\mathcal{Q}^t_{\mathcal{F}_{ns}}$ is the set of transient modes to represent a non-structural fault occurrence,
  - $\mathcal{Q}_{\mathcal{F}_{ns}}$ is the set of modes representing non-structural fault behaviours.
- $\overline{q}_0$ is the initial state,
- $\overline{\Sigma} = \Sigma \cup \Sigma^{Sig} \cup \Sigma_{\mathcal{F}_{ns}} \cup \Sigma^{Sig}_{\mathcal{F}_{ns}}$ is the set of events where:
  - Σ is the set of system events,
  - Σ^{Sig} is the set of signature-events generated by function f_{Sig_ev} defined by (7),
  - $\Sigma_{\mathcal{F}_{ns}}$ is the set of fault events related to a non-structural fault occurrence,
  - $\Sigma^{Sig}_{\mathcal{F}_{ns}}$ is the set of signature-events for non-structural faults generated by function f_{Sig_ev} defined by (7),
- $\overline{\mathcal{T}} : \overline{\mathcal{Q}} \times \overline{\Sigma} \mapsto \overline{\mathcal{Q}}$ is the partial transition function of the behaviour automaton.

In this paper, it is proposed to build B incrementally following Algorithm 1, which is an adaptation of the previous approach proposed in [Vento *et al.*, 2011]. The algorithm explores HA taking into account only the modes in which the real system is possibly operating at time instant k.

Assuming that the system is possibly operating in a given mode or set of modes denoted by q_D, to build incrementally

Algorithm 1 B_Builder(q_D)

1: Create a queue $\mathcal{L}$.
2: **for all** $q_i \in q_D$ **do**
3: Enqueue q_i onto $\mathcal{L}$
4: **end for**
5: **while** $\mathcal{L}$ is not empty **do**
6: q_i:=dequeue $\mathcal{L}$
7: **for all** $q_j \in Succs(q_i)$ **do**
8: **if** $q_j \notin \mathcal{Q} \cap \overline{\mathcal{Q}}$ **then**
9: $\overline{\mathcal{Q}} = \{q_j\} \cup \overline{\mathcal{Q}}$
10: Compute residual expression $\mathbf{r}_j(\bullet)$
11: Classify q_j into $\mathcal{Q}_{disc}$.
12: **if** q_j creates a new group ν in $\mathcal{Q}_{disc}$ **then**
13: Calculate $\mathbf{FS}_{\nu_j}$ associated to $\mathcal{F}_{ns}$
14: Determine the subsets of detectable faults $\mathcal{F}^*_{\nu_j}$.
15: Determine the set of non-detectable faults $\mathcal{F}^{'}_{\nu_j}$
16: **end if**
17: **end if**
18: Define σ such that $\mathcal{T}(q_i, \sigma) = q_j$:
19: **if** $\sigma \notin \overline{\Sigma}$ **then**
20: $\overline{\Sigma} = \{\sigma\} \cup \overline{\Sigma}$
21: **end if**
22: **if** $\sigma \in \Sigma_o$ **then**
23: $\overline{\mathcal{T}}(q_i, \sigma) := q_j$.
24: **else**
25: **if** q_i and q_j are discernible **then**
26: $\mathcal{Q}^t = \{q^t_{i-j}\} \cup \mathcal{Q}^t$
27: **if** $\delta_{\nu_i - \nu_j} \notin \Sigma^{Sig}$ **then**
28: $\Sigma^{Sig} = \{\delta_{\nu_i - \nu_j}\} \cup \Sigma^{Sig}$
29: **end if**
30: $\overline{\mathcal{T}}(q_i, \sigma) := q^t_{i-j}$.
31: $\overline{\mathcal{T}}(q^t_{i-j}, \delta_{\nu_i - \nu_j}) := q_j$.
32: **else**
33: Enqueue q_j onto $\mathcal{L}$
34: $\overline{\mathcal{T}}(q_i, \sigma) := q_j$.
35: **end if**
36: **end if**
37: **end for**
38: **for all** $f_l \in \mathcal{F}_{ns}$ **do**
39: $\overline{\mathcal{Q}} = \{q_i^{f_l}\} \cup \overline{\mathcal{Q}}$
40: **if** $\sigma_{f_l} \notin \overline{\Sigma}_{\mathcal{F}_{ns}}$ **then**
41: $\overline{\Sigma}_{\mathcal{F}_{ns}} = \{\sigma_{f_l}\} \cup \overline{\Sigma}_{\mathcal{F}_{ns}}$
42: **end if**
43: **if** $f_l \notin \mathcal{F}^{'}_{\nu_i}$ **then**
44: $\overline{\mathcal{Q}} = \{q^t_{i-f_l}\} \cup \overline{\mathcal{Q}}$
45: **if** $\delta^*_{\mathcal{F}^\iota_{\nu_i}} \notin \Sigma^{Sig}_{\mathcal{F}_{ns}}$ **then**
46: $\Sigma^{Sig}_{\mathcal{F}_{ns}} = \{\delta^*_{\mathcal{F}^\iota_{\nu_i}}\} \cup \Sigma^{Sig}_{\mathcal{F}_{ns}}$
47: **end if**
48: $\overline{\mathcal{T}}(q, \sigma_{f_l}) := q^t_{i-f_l}$.
49: $\overline{\mathcal{T}}(q^t_{i-f_l}, \delta^*_{\mathcal{F}^\iota_{\nu_i}}) := q_i^{f_l}$.
50: **else**
51: $\overline{\mathcal{T}}(q_i, \sigma_{f_l}) := q_i^{f_l}$.
52: **end if**
53: **end for**
54: **end while**

B, all the successor modes for each mode of q_D ($Succs(q_i)$) are explored.

The set of explored modes is partitioned into subsets of non discernible-modes, forming the partition denoted by $\mathcal{Q}_{disc} = \mathcal{Q}_{\nu_1} \cup \cdots \cup \mathcal{Q}_{\nu_N}$. This information is stored in a knowledge base used by Algorithm 1 such that a new set of residuals is generated only when a mode has not been previously visited.

Transitions of HA are integrated in B, evaluating the discernability property if necessary and analyzing the fault signature matrix to include non-structural faults as faulty modes. If a transition of HA associated with an observable event is found in the exploration, it is kept in B (see line 23). Otherwise, the discernability property is evaluated between these pair of modes. If the two modes are discernible then an intermediate [4] mode is added between these modes. The outgoing transition is associated a signature-event, indicating that this mode change can be observed by means of the consistency indicators (see lines from 25-31).

To include information about non-structural faults in B, lines 38 to 49 of Algorithm 1 show how a fault signature matrix is generated associated with the current (set of) modes q_D. It is then analyzed to include the non-structural faulty modes $\mathcal{Q}_{\mathcal{F}_{ns}}$ and the transient modes $\mathcal{Q}^t_{\mathcal{F}_{ns}}$ based on discernability. Analyzing this matrix, the set of non-structural faults can be partitioned into detectable ($\mathcal{F}^*_{\nu_j}$) and non-detectable ($\mathcal{F}^{'}_{\nu_j}$) fault subsets. The set of detectable faults is further partitioned into: $\mathcal{F}^*_{\nu_j} = \mathcal{F}^1_{\nu_j} \cup \cdots \cup \mathcal{F}^N_{\nu_j}$ (lines 13 to 15), where N is the number of discernible, hence isolable, fault subsets.

The signature-events can represent both a mode change or a non-structural fault occurrence, according to this the signature event can be labeled according to:

$$f_{Sig\_ev} : \mathcal{Q} \times \overline{\mathcal{Q}} \rightarrow \Sigma^{Sig} \tag{7}$$

$$f_{Sig\_ev}(q_i, q_j) \mapsto \begin{cases} \delta_{\nu_i - \nu_j} \in \Sigma^{Sig} & \text{if } (q_i, q_j) \text{ are discernible} \\ \delta^*_{\mathcal{F}^\iota_{\nu_i}} \in \Sigma^{Sig}_{\mathcal{F}_{ns}} & \text{if } \mathbf{FS}_{\nu_i}(\bullet, f_l) \neq \mathbf{0} \end{cases}$$

where $q_i \in \mathcal{Q}_{\nu_i}$ and $q_j \in \mathcal{Q}_{\nu_j}$ with $\mathcal{Q}_{\nu_i}, \mathcal{Q}_{\nu_j} \subseteq \mathcal{Q}_{disc}$, hence $\delta_{\nu_i - \nu_j}$ denotes an event associated with a mode change between modes in HA and $\delta^*_{\mathcal{F}^\iota_{\nu_i}}$ denotes an event associated with a fault f_l belonging to the set $\mathcal{F}^\iota_{\nu_i} \subseteq \mathcal{F}^*_{\nu_i}$, and $q_j \in \mathcal{Q}_{\mathcal{F}_{ns}}$ associated with the non-structural fault.

4.2 Hybrid Diagnoser

The diagnoser is a finite state machine $D =< \mathcal{Q}_D, \Sigma_D, T_D, q_{D_0} >$, where:

- $q_{D_0} = \{q_0, \emptyset\}$ is the initial state of the diagnoser, which is assumed to correspond to a nominal system mode.
- $\mathcal{Q}_D$ is the set of diagnoser states. An element $q_D \in \mathcal{Q}_D$ is a set of the form $q_D = \{(q_1, l_1), (q_2, l_2), \cdots (q_n, l_n)\}$, where $q_i \in \overline{\mathcal{Q}}$ and $l_i \in \Delta$ where Δ defines the power set of fault labels $\Delta_{\mathcal{F}} = \Delta_{\mathcal{F}_s} \cup \Delta_{\mathcal{F}_{ns}}$ with $\Delta_{\mathcal{F}_s} = \{f_1, \cdots, f_\gamma\}$, and $\Delta_{\mathcal{F}_{ns}} = \{f^*_1, \cdots, f^*_\mu\}$ respectively, $\gamma + \mu$ is the total number of faults in the system and $\gamma, \mu \in \mathcal{Z}^+$. In $\Delta_{\mathcal{F}}$, $\emptyset$ represents the nominal behaviour,
- $\Sigma_D = \overline{\Sigma}_o$ is the set of all observable events.
- $\mathcal{T}_D : \mathcal{Q}_D \times \overline{\Sigma}_o \mapsto \mathcal{Q}_D$ is the partial transition function of the diagnoser.

The transition function $\mathcal{T}_D$ can be calculated according to the propagation algorithms explained in [Sampath *et al.*, 1995], from the incremental B obtained during system mode tracking. The algorithm to build the transition function is executed after the occurrence of an observable event whenever the state has not been previously visited. The part of the diagnoser obtained takes into account only the possible successor modes and hence the transitions that can occur next.

[4] The transient mode is a way to account for the hybrid automaton HA dwell time requirement. This requirement guarantees that residuals, and hence consistency indicators, can be properly computed and that signature-events can be properly issued [Bayoudh *et al.*, 2008]

5 Application Case Study

To illustrate the method, a part of the Barcelona sewer network presented in [Vento *et al.*, 2011] is used (see Fig. 2). The elements that appear in the example are: two virtual tanks T_0 and T_1, a control gate ($gate_1$), two pluviometers P_{19} and P_{16} to measure the rain intensity and two limnimeters L_{39} and L_{41} to measure the sewer level. The control gate is commanded by a controller applying open/close gate actions depending on the flow in the sewer.

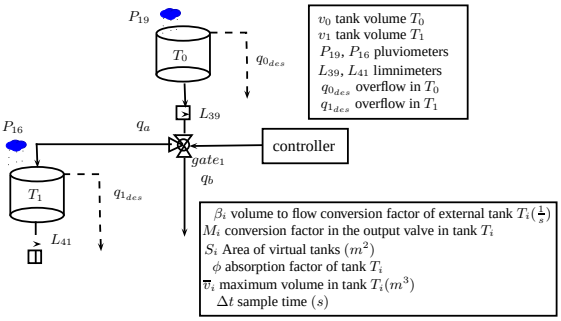
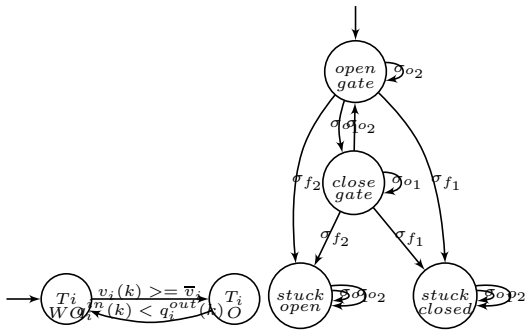


Figure 2: A small part of the sewer network

5.1 Hybrid Modeling

A hybrid automaton model can be obtained to represent the hybrid phenomena present in the network associated with the virtual tanks and the control gate. A way to obtain the hybrid model is to provide the automata for each component (T_0, T_1 and $gate_1$) and then synchronizing all automata to get the global model [Henzinger, 1996]. The automaton for a virtual tank is given by two discrete states: overflow (O) and non overflow (WO) as is shown in Fig. 3. Regarding the control gate, there are four discrete states, the nominal behaviours (open or closed) and the faulty behaviours (stuck open or stuck closed).



(a) Automaton for a virtual tank T_i (b) Automaton for a control gate

Figure 3: Automaton for the components

The global hybrid automaton has 16 operation modes modes where 8 of them correspond to nominal modes and the rest correspond to different configurations involving a control gate fault (stuck closed or stuck open). The set of structural faults is given by $\mathcal{F}_s = \{f_1, f_2\}$ and the non-structural faults correspond to additive faults in sensor ($L_{39}, L_{41}, P_{19}, P_{16}$) given by $\mathcal{F}_{ns} = \{f_3, f_4, f_5, f_6\}$, respectively.

The set $\Sigma_s = \{\sigma_{uo1}, \sigma_{uo2}, \sigma_{uo3}, \sigma_{uo4}\}$ represents the unobservable spontaneous events. Event σ_{uo1} corresponds to the volume in tank T_0 reaching its maximum $v_0 \geq \overline{v}_0$. Event σ_{uo2} corresponds to the input flow being less than the output flow from T_0 (i.e., $q_0^{in} < q_0^{out}$). The other events are related to the virtual tank T_1. The set $\Sigma_{\mathcal{F}_s} = \{\sigma_{f1}, \sigma_{f2}\}$ represents the fault events related to the structural faulty modes and $\Sigma_{\mathcal{F}_{ns}} = \{\sigma_{f2}, \sigma_{f3}, \sigma_{f4}, \sigma_{f5}\}$ the non-structural fault events. The set $\Sigma_c = \{\sigma_{o1}, \sigma_{o2}\}$ gathers input events corresponding to closing or opening the valve issued by the controller.

5.2 Simulation Results

Assume that the system tracks the mode sequence $\{q_1, q_3, q_1, q_5\}$ and the sampling time is $\Delta t = 300s$. Mode q_1 refers to the situation in which no tank is in overflow. Then, T_1 is in overflow during a period of time (mode q_3) until it leaves the overflow situation (mode q_1). Later, the control gate is closed. The diagnoser must track the right mode sequence and detect and isolate the possible faults from an incrementally built behavior automaton B.

Assuming that the initial mode is known and it is q_1, then applying Algorithms 1 the initial B is shown in Fig. 4. The initial diagnoser is obtained applying the propagation algorithm described in [Sampath *et al.*, 1995] to the initial B. Then, the diagnoser waits for the occurrence of an event. Notice that the initial B includes the possible events that may occur. These events are δ_{13}, δ_{14}, δ_{12}, $\delta^*_{\mathcal{F}^1_{\nu_1}}$, $\delta^*_{\mathcal{F}^2_{\nu_1}}$, $\delta^*_{\mathcal{F}^3_{\nu_1}}$, σ_{o_1} and σ_{o_2}.

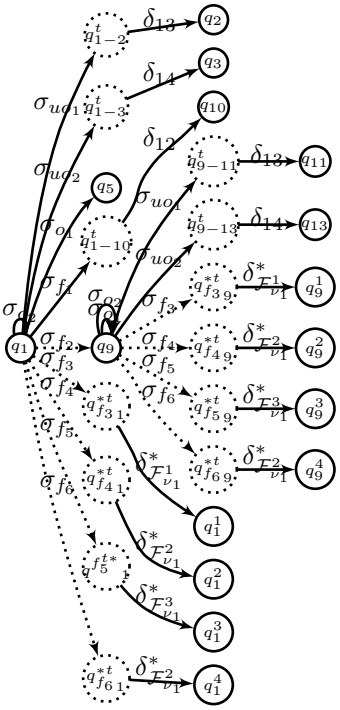


Figure 4: Initial incremental B

Notice that $\mathcal{Q}_{disc} = \mathcal{Q}_{\nu_1} \cup \mathcal{Q}_{\nu_2} \cup \mathcal{Q}_{\nu_3} \cup \mathcal{Q}_{\nu_4} \cup \mathcal{Q}_{\nu_5}$. Fig. 5 shows the value of the set of residuals for all groups in $\mathcal{Q}_{disc}$. According to the set of residuals for the set of modes of HA, two signature-events, δ_{14} and δ_{41}, were identified using the consistency indicators appropriately. These signature-events correspond to transitions $q_1 \rightarrow q_3$ and

$q_3 \rightarrow q_1$ with $q_1 \in \mathcal{Q}_{\nu_1}$ and $q_3 \in \mathcal{Q}_{\nu_4}$. Notice for instance that when the system is in mode q_3, $\Phi_{\nu_1}(k) \neq \mathbf{0}$ and $\Phi_{\nu_4}(k) = \mathbf{0}$. Both modes $q_1, q_3 \in \mathcal{Q}$ represent a nominal behaviour.

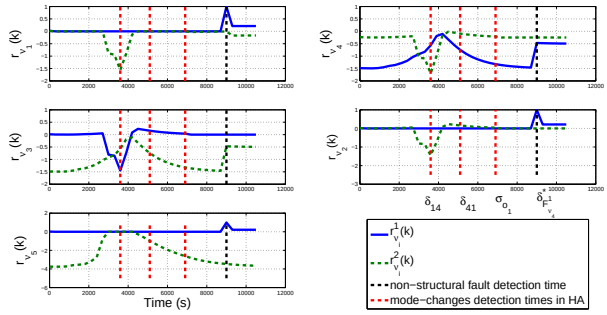


Figure 5: Residuals generation for the set of modes in HA

Later, the observable event σ_{o1} occurs, corresponding to the control gate closing. This event is identified instantaneously and indicates that a mode change from q_1 to q_5 takes place. Fig. 5 shows the set of residuals of mode q_1 and $q_5 \in \mathcal{Q}_{\nu_2}$. It should be noticed that the residuals in mode q_5 are consistent with measurements after δ_{o_1} is detected, i.e. $\Phi_{\nu_2}(k) = \mathbf{0}$.

The set of residuals are only generated for modes that are visited in HA. In this way, the efficient use of memory is guaranteed. There is a set of two residuals per group using the expression given by (4).

A non-structural fault then occurs at $9000s$ (indicated in Fig. 5 with a black vertical dashed line). Then, the diagnoser detects the fault at $9300s$. The set of consistency indicators of mode q_5 are used to isolate the fault. The observed signature is $\begin{bmatrix} 1 & 0 \end{bmatrix}^t$ which, according to $\mathbf{FS}_{\nu_4}$, corresponds to a fault in sensor L_{39}. Finally, the hybrid diagnoser stops and reports the diagnosis. Indeed, a non-structural faults needs to be repaired before the diagnoser can resume.

The report given by the hybrid diagnoser is shown in Table 1. The first column represents mode changes in HA, the second one, the identified events. The third column corresponds to the diagnoser state information and total number of states generated, the fourth one shows the total number of residuals generated. The last two columns show the occurrence time and the detection time of the identified events.

Mode change	Reported event	State diagnoser (total number of states)	Generated residuals	Occurrence time (s)	Detection time (s)
$q_1 \rightarrow q_3$	δ_{14}	$(q_3, \{\}), (q_{13}, \{f_2\})$ (7)	8	3000	3300
$q_3 \rightarrow q_1$	δ_{41}	$(q_1, \{\}), (q_9, \{f_2\})$ (13)	10	4200	4500
$q_1 \rightarrow q_5$	σ_{o_1}	$(q_5, \{\}), (q_9, \{f_2\})$ (23)	10	6900	6900
$q_5 \rightarrow q_5^2$ fault $f_3 \in \mathcal{F}_{ns}$ in Mode q_5	$\delta^*_{\mathcal{F}^1_{ns_4}}$	$(q_5^1, \{f_3\}), (q_5^2, \{f_5\})$ (23)	10	9000	9300
Total		23	10		

Table 1: Hybrid diagnoser report

Table 2 provides a comparison of the results obtained with the present method and those obtained with the off-line diagnoser generation [Vento *et al.*, 2011; Bayoudh *et al.*, 2008], standing out the benefits of the proposed method.

6 Conclusions

A method to incrementally build a hybrid diagnoser has been presented. The diagnoser is built whenever the system requires it after an event occurs (signature-event or input event). The method comprises the detection and isolation of structural and non-structural faults which are included in the system model. The diagnoser executes the tasks of mode recognition and identification using the consistency indicators generated from a set of residuals for every mode and then builds the part of the diagnoser required by the system operation. Thus, the diagnoser obtained requires less memory space and can be efficiently obtained online. An illustrative example of the proposed approach based on a piece of the Barcelona sewer network is used. Future work will consider to add the incremental design of the HA from the component hybrid automata. This will nicely complete the proposed incremental approach, avoiding not only to store the whole diagnoser but also the whole hybrid model.

	Previous methods	Proposed method
Number of diagnoser states	75	23
Number of residuals generated	10	10
Computational complexity	Exponential $(2^{N_{states_D}})$ N_{states_D} Total number of diagnoser state	Lineal $(N_{Succs(q_D)})$ $N_{Succs(q_D)}$ Total number of successors

Table 2: Comparison with previous methods according to the simulation scenario

References

[Bayoudh *et al.*, 2008] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Hybrid systems diagnosis by coupling continuous and discrete event techniques. In *Proceedings of the 17th International Federation of Automatic Control, World Congress, IFAC-WC*, pages 7265–7270, Seoul (Korea), 2008.

[Daigle, 2008] M. Daigle. *A Qualitative Event-Based Approach to Fault Diagnosis of Hybrid Systems*. PhD thesis, Faculty of the Graduate School of Vanderbilt University, Nashville, Tennessee, 2008.

[Ding *et al.*, 2008] X. Ding, M. Kinnaert, J. Lunze, and M. Staroswiecki. *Model Based Fault Diagnosis Techniques*. Springer, 2008.

[Henzinger, 1996] T. Henzinger. The theory of hybrid automata. *Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 278–292, July 1996.

[Lygeros *et al.*, 2003] J. Lygeros, K. Henrik, and J. Zhang. Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48, January 2003.

[Mezyani, 2007] T. Mezyani. *Diagnostic des Systèmes Dynamiques Hybrides*. PhD thesis, Université Lille1, France, 2007.

[Narasimhan and Biswas, 2007] S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *IEEE Transactions on Systems, Man and Cybernetics*, 37(3), May 2007.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, and S. Lafortune. Diagnosability of discrete-event system. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[Vento *et al.*, 2011] J. Vento, V. Puig, and R. Sarrate. A methodology for building a fault diagnoser for hybrid systems. In *9th European Workshop on Advance Control and Diagnosis*, Budapest, Hungry, November 2011.

Preliminaries On Complexity of Diagnosis of Discrete-Event Systems

Gianfranco Lamperti and **Marina Zanella**
Department of Information Engineering, University of Brescia, Italy
email: {gianfranco.lamperti,marina.zanella}@ing.unibs.it

Abstract

This paper is an attempt to move a small step toward a complexity analysis of diagnosis of discrete-event systems (DESs). The considered conceptual model of the behavior of DESs are automata. If the DES is distributed, its behavior is implicitly described by the automaton resulting from the synchronous composition of the component automata. Solving a diagnosis problem inherent to a DES amounts to performing a search within the relevant automaton so as to find a behavioral path that starts from a given state and generates a given observation. Three dimensions are taken into account: temporal and logical uncertainty of the observation, and uncertainty about the initial state. The main outcome of this preliminary analysis is that checking whether there exists any solution to a given diagnosis problem is in PSPACE. If such a check is inherent to a non-distributed DES, it is in NP when the observation is both temporally and logically certain, while, if the DES is distributed, it is NP-hard, whichever the observation.

1 Introduction

A general definition of a DES [Cassandras and Lafortune, 2008] reads 'a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time'. At an untimed abstraction level, a DES is described by a language. An automaton [Hopcroft *et al.*, 2006] is the most intuitive model to represent a language. This is the reason why automata were adopted as modeling primitives since the very beginning of research on Model-Based Diagnosis (MBD) of DESs [Sampath *et al.*, 1995] in the middle '90s, thus leading to what in [Grastien *et al.*, 2007] are called the 'classical' approaches.

So many years after, in spite of meaningful results on the complexity of diagnosability of DESs represented as untimed automata [Jiang *et al.*, 2001; Rintanen, 2007], a systematic analysis of the complexity of diagnosis is still missing. A DES diagnosis problem consists of a DES (automaton), the initial state of the DES and an observation inherent to the DES (assuming that some state changes of the DES are observable). Solving it means finding all the paths in the given automaton that may have produced the given observation. In [Grastien *et al.*, 2007] a diagnosis problem inherent to a DES, given an initial state that is completely certain, is first presented as a path finding problem, and then formulated as a SAT problem. This looks like a reduction of the diagnosis problem to a known NP-complete problem. Such a reduction, however, does not allow us to draw any conclusion on the complexity of diagnosis of DESs, neither when the temporal order of the observed events is certain or uncertain nor when the values of observed events are uncertain (a case which is not encompassed in [Grastien *et al.*, 2007]). Moreover, the same paper [Grastien *et al.*, 2007] states that 'diagnosis by SAT' is 'a problem harder in general than' the problem dealt with by 'the classical algorithms', which raises further questions about the complexity of diagnosis of DESs. Finding the paths (even just one of them) consistent with the given observation is at least as hard as deciding whether any such a path exists. In other words, the complexity of the diagnosis existence problem is a lower bound of the complexity of the diagnosis problem.

In this paper, before proving some theorems as to the complexity of the DES diagnosis existence problem (Section 3), the (automata based) conceptual model adopted in the analysis is described (Section 2). Conclusions are drawn in Section 4.

2 Conceptual model

A MBD *problem instance* inherent to a dynamic system modeled as a DES is a triple $(\Sigma, \Sigma_0, \mathcal{O})$, where Σ is the *DES model*, Σ_0 is the *initial state* of Σ, representing the state of the considered system at the start of a (finite) time interval of interest, and $\mathcal{O}$ is the (finite) *observation* of the behavior of the system over such a time interval.

2.1 Discrete-event system

The behavior of a DES is represented as a finite automaton (FA) Σ. In a conceptual model aimed at complexity analysis, we are interested only in the set S of *states* and the set T of state *transitions* of such an FA, where the unique identifier of each transition is accompanied by the source and target states of the transition itself. Let $\gamma : T \to E_{obs}$ be a partial *observation function*, where E_{obs} is a finite set (alphabet) of *observable events*; if γ is defined for $t \in T$, then t is *observable*. Fig. 1 displays the model of a DES consisting of 11 states (numbered from 0 to 10) and 21 transitions (whose identifiers range from $t1$ to $t21$), 10 of which are observable (each observable transition is labeled with the relevant observable event).

The initial state Σ_0 is a subset of S, that is, $\Sigma_0 \subseteq S$, which means that Σ, at the beginning of the time interval of interest, may be in any state in Σ_0 [Sohrabi *et al.*, 2010]. If Σ_0 is a singleton, then the initial state of Σ is *certain*, otherwise it is *uncertain*.

A DES is *distributed* if it consists of several interacting *components*, each of which is a DES itself. If the identifier of a transition belongs to the T set of several components, then such a transition occurs only if and when it can occur simultaneously in all the components that share it, that is, it is an outgoing transition of all the current states of all the components that share it. Therefore, the FA relevant

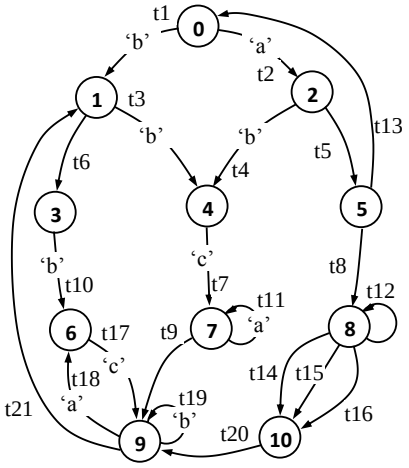


Figure 1: DES model.

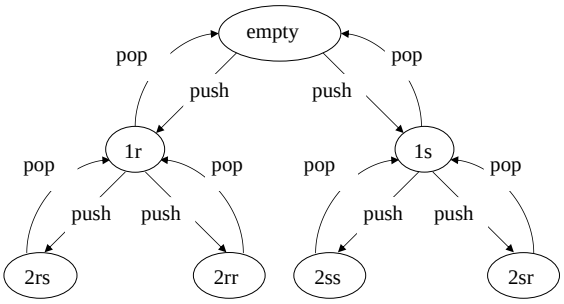


Figure 2: Component model of a link.

to the whole system is the one (implicitly[1]) resulting from the parallel composition (often called synchronous composition [Cassandras and Lafortune, 2008]) of all the component FAs, where such a synchronization is based on shared transitions, which are in fact called *synchronous* transitions.

There are approaches in the literature [Lamperti and Zanella, 2003] where components interact 'asynchronously', by exchanging (communication) events over *links*. Including links in a DES model is a matter of expressive power, not of computational power. A link, whichever its capacity and its policy (FIFO, LIFO, etc.), could be represented as a component itself, which shares transitions with both the component(s) that feed(s) the link and the component(s) that extract(s) events from the link. Fig. 2 shows the component model of a link whose capacity is 2, whose policy is LIFO, where the events sent on the link are of two distinct types (say r and s). Each downward arrow represents a transition (shared with a component that feeds the link) that pushes an event on the stack of events in the link, while each upward arrow represents a transition (shared with a component that is fed by the link) that extracts the event on the top of the stack. State *empty* represents the case when the link is empty, while each state whose identifier begins with 1 or 2 represents the case when the link contains one or two events, respectively. The string of characters following the digit recalls the content of the stack.

Specific primitives for automata communication were introduced for several reasons: first of all, the link primitive is generic (it specifies just the capacity and the policy), that is, independent of the number and types of exchanged events, while the component model of a link depends on them; second, the size of the component model of a link includes a number of states that equals $\sum_{i=0}^{n} d^i = (d^{n+1}-1)/(d-1)$, where $d > 1$ is the cardinality of the set of communication events and n is the capacity of the link; third, the link primitive is processed more efficiently (by ad hoc algorithms) than the component model of a link. However, the class of DESs described by using links is a subset of the class of DESs described as interacting synchronously since an asynchronous communication can be modeled through the primitive for synchronous communication.

In order to base the complexity analysis on the most general class of systems, in the following we will assume to deal with DESs conceptually modeled as synchronous systems.

[1]The FA relevant to a distributed DES is implicit since no state-of-the-art approach to MBD of DESs generates it explicitly.

2.2 Observation

The observation $\mathcal{O}$ is what has externally been perceived about the system behavior over the time interval of interest. Typically the dynamic system has undergone an evolution which corresponds to a sequence of state transitions of the DES model, starting from a state in Σ_0. Such an evolution has generated a (possibly empty) sequence of observable events belonging to E_{obs}. However, in the observation the temporal order of the emission of such events may be disrupted into a partial order or even a totally unknown order (*temporal uncertainty*). This uncertain order is compatible with several sequences of observable events, among which there is the one actually emitted by the system. Moreover, also the observable events that have taken place in the DES may have been distorted, that is, a received value may range over a finite set of possible values, among which there is the right one (*logical uncertainty*). Thus, the emitted sequence of observable events may fall into a set of sequences, denoted $||\mathcal{O}||$. Set $||\mathcal{O}||$ is finite since *(i)* the number of events in $\mathcal{O}$, denoted $|\mathcal{O}|$, is finite; *(ii)* each event ranges over a finite set of values, whose cardinality has $|E_{obs}|$ as an upper bound; and *(iii)* the number of total orders compatible with the given partial order is finite, an upper bound being $|\mathcal{O}|!$.

The model of the observation proposed in [Lamperti and Zanella, 2002] is a directed acyclic graph, where each node represents an event perceived by the observer and each arc represents a temporal precedence relationship according to the emission order. The relative emission order of any pair of nodes such that one is not reachable from the other, as the shaded ones in the observation in the center of Fig. 3, is unknown. In case the observation is logically certain, every node contains a single event; if, instead, the observation is logically uncertain, then one node at least contains several events, just one of which was actually emitted by the system. In case the observed event represented by a node of the observation graph may be just pure noise, then the node is bound to include several observable events one of which is the null event (ϵ). All the three observations in Fig. 3 are logically uncertain, and they all include a node containing the null event. Considering the observation on the left of the figure, set $||\mathcal{O}||$ includes 32 sequences, among which the following: $\langle a,b,a,a,b,c,a\rangle$, $\langle b,b,a,a,b,c,a\rangle$, $\langle a,b,c,a,b,c,a\rangle$, $\langle a,b,c,c,c,a\rangle$, $\langle b,b,c,c,c,a\rangle$.

2.3 Diagnostic output

The solution of a problem instance $(\Sigma, \Sigma_0, \mathcal{O})$ is a (possibly empty) set of *candidate diagnoses*, each of which explains what has been observed. The most concrete notion of a candidate diagnosis is that providing the information for following a path in the FA representation of the behavior of the

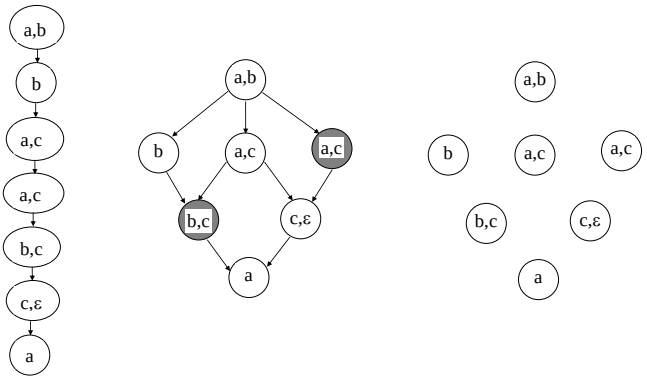


Figure 3: A totally temporally ordered observation (left), a partially temporally ordered observation (center), and an observation with unknown temporal order (right).

whole system, where such a path starts from a state in Σ_0 and generates a sequence of observable events that belongs to $||\mathcal{O}||$. This means that there exists a sequence of all the nodes in $\mathcal{O}$, where the total order of such a sequence is compliant with the (possibly partial or unknown) temporal order represented in the observation graph, such that, by picking up an event from each node of the sequence of nodes, the obtained sequence of observable events, once all the occurrences of the null event have been removed, equals the sequence of observable events generated by the path. For instance, path $\wp = \langle t2, t4, t7, t11, t9, t19, t18\rangle$ in the DES model of Fig. 1 generates the sequence of observable events $\langle a, b, c, a, b, a\rangle$: such a sequence is compliant with all the three observations in Fig. 3. A candidate diagnosis can be represented as a finite sequence of (component) transitions (where each transition uniquely singles out both the source and target component state(s) of the transition itself). A path starting from a state in Σ_0 is usually called a *history*. Thus, if state 0 belongs to Σ_0, $\wp$ is a history.

If the system model includes unobservable cycles, a set of candidate diagnoses that produce the same sequence of observable events and differ just for the number of times each unobservable cycle is followed, can be expressed at a higher abstraction level as a *route*. Syntactically, a route can be represented as a sequence of transitions (starting from a state $\sigma_0 \in \Sigma_0$) that possibly includes pairs of (possibly nested) parentheses, where each pair contains a sequence of transitions identifying an unobservable cycle.

For instance, $r = \langle t2, t5,\ t8, (t12),\ t15, t20, t19,\ t18,\ t17, t19, t18\rangle$ is a route of the DES model in Fig. 1: the pair of parentheses identifies the unobservable cycle consisting just of transition $t12$. Route r cumulatively represents an unbound number of histories, which differ from each other for the (possibly null) number of times transition $t12$ is performed, each of which, however, generates the sequence of observable events $\langle a, b, a, c, b, a\rangle$, that belongs to $||\mathcal{O}||$ for all the three observations in Fig. 3.

Some approaches [Pencolé and Cordier, 2005; Grastien *et al.*, 2007] represent (and compute) a candidate diagnosis as a sequence of sets of non interfering transitions, where two distinct transitions belonging to two distinct components do not interfere if they may occur concurrently. A set of non interfering transitions, called *trace* in [Pencolé and Cordier, 2005], is a concise representation of a 'diamond' in the system model, i.e. a portion of the FA that starts in a state, ends in another, and such that each permutation of the transitions in the set is a path from the former to the latter. This is another concise way to represent a set of histories. Actually, the advantages of both routes and traces could be combined in order to represent a till larger set of histories. However, the concepts of route and trace (and their combination) do not alter the fact that a diagnosis problem has a solution if and only if there exists a history of Σ that generates a sequence of observable events belonging to $||\mathcal{O}||$. Given a (finite, possibly empty) observation $\mathcal{O}$, a history that is consistent with it may be infinite, owing to unobservable cycles, therefore no diagnostic algorithm can produce it. The route corresponding to an infinite history, instead, is finite, therefore we will perform a decidability analysis by appealing to the notion of a route as a diagnostic output.

3 Decidability and complexity

Let D be the set of all DES diagnosis problem instances, as described in Section 2, each of which may be affected by three orthogonal forms of uncertainty: (i) temporal uncertainty in the observation, (ii) logical uncertainty in the observation, and (iii) uncertainty about the initial state. Domain D can be partitioned into D_{nd} and D_d, these including all the problem instances inherent to non-distributed and distributed DESs, respectively. Let DIAGNOSIS-EXISTENCE(D) be the set (i.e. the language) of all problem instances $P \in D$ such that $P = (\Sigma, \Sigma_0, \mathcal{O})$ is solvable, that is, there exists a route of Σ, starting from a state in Σ_0, that generates a sequence o of observable events s.t. $o \in ||\mathcal{O}||$. Without loss of generality, we assume that the last transition in a route that solves the problem is observable.

Theorem 1. *DIAGNOSIS-EXISTENCE(D) is decidable.*

Proof. (*Sketch*) The model of Σ consists of either one or several FAs, one for each component. Whichever the chosen initial states of all components, the synchronous composition of all such FAs is an FA, which includes a finite number of unobservable cycles (if any). For whichever observation, the number of nodes is finite, and each node contains a finite set of observable events. Thus, set $||\mathcal{O}||$, which contains completely certain (finite) sequences of observable events, is finite. So it is possible, first of all, to generate all the sequences in $||\mathcal{O}||$. Then, for each state $\sigma_0 \in \Sigma_0$, the synchronous composition of all the FAs starting from σ_0 can be carried out, thus obtaining an FA, say A_{σ_0}. Finally, for each sequence $o \in ||\mathcal{O}||$, a brute force search within A_{σ_0} can be performed to find out whether a route starting from σ_0 and generating o exists. Such a search requires to detect possible unobservable cycles occurring along a path, so as not to visit any of them an unbound number of times: since the number and length of such cycles is finite, the brute force algorithm can detect them. As soon as a route that starts from σ_0 and generates o is found, 'yes' is returned. If no such route is found, 'no' is returned. □

Let us call *minimal history* a (necessarily finite) history drawn from a route by removing all unobservable cycles.

Theorem 2. *A DES diagnosis problem $P = (\Sigma, \Sigma_0, \mathcal{O})$, $P \in D$, is solvable iff there exists a minimal history that solves it.*

Proof. By definition, P is solvable iff there exists a route that solves it. If such a route does not include any unobservable cycles, then it is a minimal history. If the route includes some unobservable cycles, then a minimal history can be drawn from it. Such a history solves the problem as well since all its transitions are executable in the given order and produce the same observation as the route. □

Theorem 3. *Every minimal history h is such that, for each pair (o_i, o_{i+1}) of consecutive observable events produced by h, the subsequence of unobservable transitions of h between the two observable transitions that produce o_i and o_{i+1}, respectively, cannot include more than $L_{max} - 1$ transitions, where L_{max} is the number of states of Σ.*

Proof. Let us assume that h produces a sequence o of observable events. Each state reached by Σ according to h, here called *history-state*, is univocally identified by a pair consisting in the state of Σ and an *observation index* $i \in [0, \ldots, |o|]$ that specifies the (sub)sequence of the observable events in o that have already taken place. Thus, value 0 of index i denotes that no observable events have taken place, while value $|o|$ denotes that all observable events in o have taken place. Given a history-state characterized by value i of the observation index, a history-state characterized by value $i + 1$ is reachable from it iff one observable transition is performed, where such a transition produces observable event o_{i+1}. However, before reaching the new history-state, several unobservable transitions may be performed, these leading from the current state to other history-states characterized by the same value i of the observation index. The maximum number of history-states characterized by value i and distinct from the current one[2] is $L_{max} - 1$. □

Corollary 1. *The length of a minimal history solving a diagnosis problem $P = (\Sigma, \Sigma_0, \mathcal{O})$, $P \in D$, is $O(|\mathcal{O}|L_{max})$, where $|\mathcal{O}|$ is the number of nodes in $\mathcal{O}$ and L_{max} is the number of states of Σ.*

Proof. Since the considered history solves the problem, it produces a sequence $o \in ||\mathcal{O}||$, whose maximum length is $|\mathcal{O}|$. As stated by Theorem 3, in a minimal history the number of unobservable transitions in between two observable ones cannot exceed $L_{max} - 1$. Therefore the length of a minimal history solving the problem is $O(|\mathcal{O}|L_{max})$. □

Theorem 4. *DIAGNOSIS-EXISTENCE(D_{nd}) is in NP if the observation is both logically and temporally certain.*

Proof. Given $P \in D_{nd}$, let the certificate be a minimal history h. Based on Corollary 1, $|h|$ is polynomial in the size of P. The pseudo-code of a verification algorithm which is polynomial in time is here below.

```
1.  function Verify(P, h)
2.    where P = (Σ, Σ0, O): a problem instance in D_nd,
3.    Σ0: a set of initial states,
4.    O = ⟨e1, ..., e|O|⟩: a sequence of observable events,
5.    and h = ⟨t1, ..., t|h|⟩: a minimal history of Σ.

6.    for each σ0 ∈ Σ0 do
7.      s = σ0
8.      j = 1
9.      for each i ∈ [1 .. |h|] do
10.       if t_i does not exit from s then
11.         go to the next iteration of cycle at line 6
12.       if t_i is observable then
13.         e = γ(t_i) // observable event generated by t_i
14.         if j ≤ |O| and e == e_j then j++ else
15.           go to the next iteration of cycle at line 6
16.       s = state reachable from s by applying t_i
17.     end-for
18.     if j == |O| + 1 then return true
19.   end-for
20.   return false
21. end {Verify}
```

For each possible initial state in Σ_0, *Verify* simply takes into account each transition t_i in h, according to the order transitions appear in h. It checks whether t_i exits for the current state, and, in case t_i is observable, whether it produces the observable event expected according to $\mathcal{O}$. If, after all transitions in h have been encompassed, all (and only) the observable events in $\mathcal{O}$ have been produced, then true is returned, meaning that h actually solves P. □

[2]Only history-states distinct from the current one have to be considered as h is a minimal history and, as such, it does not include any unobservable cycle. A sequence of transitions that comes back to the current state without any observable event having occurred would instead follow an unobservable cycle.

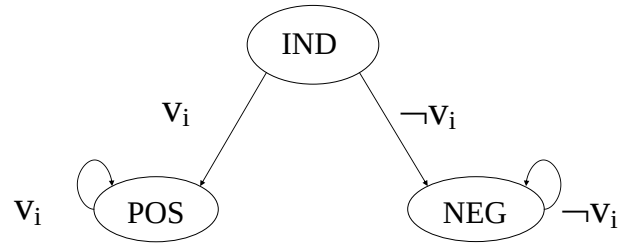


Figure 4: Automaton corresponding to a Boolean variable.

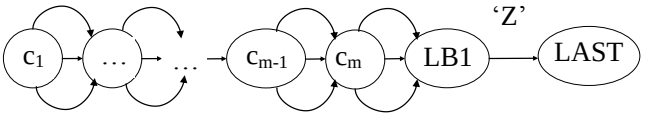


Figure 5: Automaton corresponding to a 3-CNF.

Theorem 5. *DIAGNOSIS-EXISTENCE(D_d) is NP-hard.*

Proof. NP-hardness is proven by means of a reduction from 3-SAT. Let ϕ be a Boolean formula in 3-CNF with variables $V = \{v_1, ..., v_n\}$ and clauses $C = \{c_1, ..., c_m\}$. We will construct an instance of DIAGNOSIS-EXISTENCE(D_d) relevant to a diagnosis problem $P_\phi = (\Sigma_\phi, \Sigma_{\phi 0}, \mathcal{O}_\phi)$ such that P_ϕ is solvable iff ϕ is satisfiable. Since a completely certain observation is a special case of a (temporally and/or logically) uncertain observation, and a completely certain initial state is a special case of an uncertain one, we will propose a reduction where P_ϕ is not affected by any uncertainty, since the same reduction can be reused in all the other settings.

(*Reduction*) For each $v_i \in V$ we build an FA resembling that in Fig. 4, where each transition is marked by a tag for synchronization. Then we build an FA corresponding to C, depicted in Fig. 5, this way: it includes $m + 2$ states, where each of the first m states corresponds to a distinct clause c_i. The generic state corresponding to clause c_i has three exiting transitions, each having a synchronization tag, corresponding to a distinct literal of c_i. Informally, we can move from the state corresponding to clause c_i to that corresponding to clause c_{i+1} only if c_i is true, which holds only iff at least one literal of its is true. This means that, when the current state of the FA in Fig. 5 is c_{i+1}, all clauses $c_1, \ldots, c_i$ are true. State $LB1$ has an exiting transition, generating observable event Z and leading to state *LAST*. Informally, state $LB1$ is reached only if all clauses are true. This is the only observable transition of Σ_ϕ. Altogether, the $n + 1$ automata obtained this way constitute Σ_ϕ. The initial state $\Sigma_{\phi 0}$ includes the state of the FA in Fig. 5 corresponding to clause c_1 and the state *IND* of the remaining n automata. Observation $\mathcal{O}_\phi$ consists of just one node, containing event Z. It it easy to see that the construction of P_ϕ is polynomial in the size of ϕ.

(*Direct implication*) Let us assume that ϕ is satisfiable, that is, there exists a variable assignment $\bar{V}$ of V such that ϕ is true. Let us build a sequence h of transitions of Σ_ϕ of length $m + 1$ this way: for each clause c_i, in the order of i from 1 to m, we pick up one of its literals, and consider the assignment of the variable such a literal refers to: if the literal is v_j and the assignment inherent to v_j in $\bar{V}$ is 1, or

the literal is $\neg v_j$ and the assignment inherent to v_j in $\bar{V}$ is 0, then we add a transition marked by the tag corresponding to such a literal (that is, tag v_j for literal v_j, and tag $\neg v_j$ for literal $\neg v_j$), which corresponds to following a (synchronous) transition marked by such a tag both in the FA corresponding to variable v_j and the FA corresponding to C; otherwise, we try the next literal in c_i, until the condition is fulfilled (the condition is bound to be fulfilled by a literal of c_i since ϕ is satisfiable, henceforth all its disjunctive clauses are true). After these m transitions, we add the last transition of the sequence, that leading from state *LB1* to *LAST* in the FA corresponding to C. The obtained sequence h is a history of Σ_ϕ. In fact, starting from the initial state of each FA corresponding to a variable v_j, in a mutually exclusive way we follow either the transition marked by tag v_j (if the assignment in $\bar{V}$ is $v_j = 1$) or the transition marked by tag $\neg v_j$ (if the assignment in $\bar{V}$ is $v_j = 0$); any subsequent transition in h taken from the same FA, is marked by the same tag as the previous one (as there is a unique assignment to v_j), which is compliant with the FA itself. Moreover, such a history is compliant with $\mathcal{O}_\phi$ since its last transition produces event Z. The history is minimal since the FA in Fig. 5 forces a state change every time a transition is triggered.

(*Reverse implication*) Let h_ϕ be a history of Σ_ϕ consistent with $\mathcal{O}_\phi$, and let s_f be the state reachable by applying h_ϕ to $\Sigma_{\phi 0}$. Since the only observable transition of Σ_ϕ is that leading from state *LB1* to *LAST* in the FA in Fig. 5, such a transition has necessarily to belong to h_ϕ and that it has to be preceded by m (unobservable synchronous) transitions that lead from the initial state c_1 to state $LB1$. Since every FA corresponding to a variable v_i (Fig. 4) contains just transitions that are synchronous with some transitions in the FA corresponding to C (Fig. 5), in h_ϕ there cannot be more that m transitions preceding the only observable one. There cannot be any transitions following the observable one since there are no transitions exiting from state *LAST* in the FA in Fig. 5. Moreover, by construction (see Fig. 4), h_ϕ cannot include any pair of transitions that belong to the same FA corresponding to a variable in V and are marked by distinct events. Let us consider an assignment of V obtained this way: for each variable $v_j \in V$, if the state of the FA relevant to v_j in s_f is *POS*, then the assignment is $v_j = 1$; if the state of the FA relevant to v_j in s_f is *NEG*, then the assignment is $v_j = 0$; finally, if the state of the FA relevant to v_j in s_f is *IND*, then the assignment is indifferently either $v_j = 1$ or $v_j = 0$. This assignment satisfies ϕ, that is, each clause c_i in C is true, since there exists a transition exiting from the i-th state of the FA corresponding to C, where such a transition corresponds to a literal in clause c_i, which is the same as that of the i-th transition in h_ϕ. □

No upper bound has been given for the complexity of DIAGNOSIS-EXISTENCE. The next theorem, which is inherent to both distributed and non-distributed DESs, is all we know so far.

Theorem 6. *DIAGNOSIS-EXISTENCE(D) is in PSPACE.*

Proof. We show an algorithm, called *Existence*, that solves DIAGNOSIS-EXISTENCE(D) in polynomial space. The notions of history-state and observation index, defined in the proof of Theorem 3, are exploited here.

```
1.  function Existence(P)
2.    where P = (Σ, Σ_0, O): a problem instance.

3.    for each σ_0 ∈ Σ_0 do
4.      S_in = (σ_0, 0)
5.      Enumerate all sequences o ∈ ||O|| (stripped of any ε)
6.      for each o do
7.        if Check(S_in, o, 1)
8.          then return true
9.    return false
10. end {Existence}

1.  function Check(S_0, o, i)
2.    where S_0: a history-state of system Σ,
3.    o: a sequence of observable events,
4.    i: an integer value in [1 .. |o| + 1] which
        is the successor of the observation index of S_0.

5.    if i == |o| + 1
6.      then return true
7.    Enumerate all history-states S_i whose observation
      index is i
8.    for each S_i do
9.      if Path(S_0, S_i, L_max)
10.       then if Check(S_i, o, i + 1)
11.         then return true
12.   return false
13. end {Check}

1.  function Path(S_a, S_b, L)
2.    where S_a, S_b: history-states of Σ,
3.    L: an integer value in [1 .. L_max].

4.    i_a = the observation index of S_a
5.    i_b = the observation index of S_b
6.    if (i_a == i_b) ∧ there is an unobservable transition
      from S_a to S_b
7.      then return true
8.      else if (i_b > i_a) ∧ there is an observable transition
        from S_a to S_b ∧ such a transition generates o[i_b]
9.      then return true
10.     else if L > 1
11.       Enumerate all history-states S' having the same
          observation index as S_a
12.       for each S' ≠ S_a do
13.         if Path(S_a, S', ⌈L/2⌉)
14.           then if Path(S', S_b, ⌈L/2⌉)
15.             then return true
16.   return false
17. end {Path}
```

Function *Existence* has to find out whether there exists a minimal history that solves problem P. Intuitively, it tries all the sequences of transitions that satisfy Theorem 3. For each state $\sigma_o \in \Sigma_o$, it creates the relevant history-state S_{in} (line 4). Then it enumerates, one by one, by using, for instance, a counter, all the sequences (of observable events) belonging to $||\mathcal{O}||$. For each of such sequences o, by invoking function *Check*, it checks whether there exists a behavioral path starting from the initial history-state S_{in} and compliant with $o[1 .. |o|]$. This means that the call of function *Check* at line 7 returns true if there exists a whole history of Σ, starting from S_{in} and producing o. In such a case, function *Existence* signals that there exists a solution of problem P by returning true. If, instead, there exists no $o \in ||\mathcal{O}||$ that can be produced by any history of Σ starting from some state in Σ_o, then *Existence* returns false.

Function *Check* is for checking whether there exists a behavioral path of Σ, starting from history-state S_0 and producing the observation $o[i .. |o|]$. If i equals $|o| + 1$, then true is returned since there certainly exists an (empty) path that does not generate any observable event. Otherwise, the search is split in two parts: one for a path that generates event $o[i]$ only (line 9), and the other for a path that generates the remaining of o, that is, $o[i + 1 .. |o|]$ (line 10). Since a path that generates event $o[i]$ ends in a history-state characterized by value i of the observation index, all the history-states S_i of Σ whose observation index value is i are enumerated, and, for each of them, by invoking func-

tion *Path* (line 9), it is tried whether there exists a path from S_0 to S_i, whose length is L_{max} at most, in accordance with Theorem 3. If such a path exists, then it is checked, by a recursive call of *Check* (line 10), whether there exists a behavioral path starting from S_i that generates the remaining of o. If this is the case, then we can conclude that there exists a behavioral path of Σ, starting from S_0 and producing the observation $o[i\,..\,|o|]$, thus *Check* returns true. If there is no path like that for any S_i, then *Check* returns false.

Function *Path* has to find out whether there exists a path, of length L at most, originating from the given history-state S_a of Σ and ending in the given history-state S_b. If there exists a transition of Σ leading from S_a to S_b, then true is returned. Otherwise, if L equals 1, false is returned. If, instead, L is greater than 1, the problem is split in two subproblems, which are conquered by recursive calls of *Path* (lines 13 and 14). Each subproblem is inherent to a path whose length is $\lceil L/2 \rceil$ at most. The first path has to start from S_a and end in a state S', which is distinct from S_a but has the same observation index as S_a. This is justified by considering that two cases can occur: either (i) *Path* is invoked by *Check*, or (ii) the call of *Path* is recursive. In the former case, the observation indexes of S_a and S_b have two consecutive values, thus the last transition in the path from S_a to S_b has to produce an observable event, while all the previous ones have to be unobservable. Hence, the first path has to lead to a state that share the same observation index as S_a. In the latter case, that is, every time *Path* is recursively called at line 13, S_a and S' have the same observation index. Every time *Path* is recursively called at line 14, either the observation indexes of S' and S'_b are the same or the latter is the successor of the former. In both cases, at any call, all the history-states having the same observation index as the first parameter of *Path* are enumerated at line 11. For each of them, if the first path exists, the existence of a path from S' to S_b is investigated. If both paths exist, true is returned. If, for any S', no such pair of paths exists, then false is returned.

Function *Path* needs to manipulate history-states, each of which includes an (integer) observation index and a state of Σ, which in turn includes the states of all components of Σ. The activation record of *Path* is $O(|\mathcal{C}|))$, where $\mathcal{C}$ is the set of all components. The length of the recursive chain of *Path* is $O(log(L))$, thus the space required by a run of *Path* is $O(|\mathcal{C}|log(L))$. The maximum value of parameter L is that passed by $Check$, that is, L_{max}, this being the product of the number of states of all components. Therefore, the space required by the run of *Path* is polynomial.

The activation record of *Check* is $O(|\mathcal{C}|))$. The length of the recursive chain of *Check* is $O(|o|)$, whose upper bound is the number of observation nodes $|\mathcal{O}|$. *Check* invokes *Path*, therefore it requires a space $O(|\mathcal{O}||\mathcal{C}|)$ + $O(|\mathcal{C}|log(L_{max}))$, which is polynomial.

Function *Existence* is not recursive and requires both the space for running *Check*, the space for variable o (which is $O(|\mathcal{O}|)$, and the space to save problem instance P. Thus, altogether, the algorithm requires a polynomial space. □

4 Conclusion

The complexity analysis faced in this paper is inherent to a decision problem (is there a candidate that solves a DES diagnosis problem?), whose complexity is a lower bound of that of the corresponding function problem (find the candidates of a DES diagnosis problem).

In [Portinale *et al.*, 2004] the decision problem consisting in determining whether an instance of a static diagnostic problem has a solution is proven to be NP-complete. Intuitively, the complexity of the same problem when dynamic systems are considered cannot be lower than that of static ones. However, the preliminary analysis performed by this paper is not enough to draw such a conclusion.

According to Theorem 4, checking the existence of a candidate is in NP if the DES is non-distributed and the observation is certain. This result is obtained since the length of a minimal history that solves the problem is polynomial in the number of states of the non-distributed DES, that is, for the effort to synchronize the behavior of system components is not included in the analysis. According to Theorem 5, checking the existence of a candidate is NP-hard if the DES is distributed. The main outcome (Theorem 6) is that deciding whether a DES diagnosis problem is solvable is in PSPACE. Future research will concentrate on finding a more precise upper bound for the complexity of such a problem, as well as on providing results inherent to non-distributed DESs in settings where the observation is uncertain.

References

[Cassandras and Lafortune, 2008] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer Science+Business Media, LLC, New York, NY, second edition, 2008.

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Modeling and solving diagnosis of discrete-event systems via satisfiability. In *Eighteenth International Workshop on Principles of Diagnosis – DX'07*, pages 114–121, Nashville, TN, 2007.

[Hopcroft *et al.*, 2006] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, third edition, 2006.

[Jiang *et al.*, 2001] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[Lamperti and Zanella, 2002] G. Lamperti and M. Zanella. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, 137(1–2):91–163, 2002.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of Active Systems – Principles and Techniques*, volume 741 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publisher, Dordrecht, NL, 2003.

[Pencolé and Cordier, 2005] Y. Pencolé and M.O. Cordier. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170, 2005.

[Portinale *et al.*, 2004] L. Portinale, D. Magro, and P. Torasso. Multi-modal diagnosis combining case-based and model-based reasoning: a formal and experimental analysis. *Artificial Intelligence*, 158(1):109–153, 2004.

[Rintanen, 2007] J. Rintanen. Diagnosers and diagnosability of succinct transition systems. In *20th International Joint Conference on Artificial Intelligence – IJCAI'07*, pages 538–544, Hyderabad, India, 2007.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[Sohrabi *et al.*, 2010] S. Sohrabi, J.A. Baier, and S. McIlraith. Diagnosis as planning revisited. In *Twelfth International Conference on Knowledge Representation and Reasoning – KR 2010*, pages 26–36, Toronto, Canada, 2010. Association for the Advancement of Artificial Intelligence.

Improving Robustness of Task Execution Against External Faults Using Simulation Based Approach

Anastassia Küstenmacher[1] and **Paul G. Plöger**[1] and **Gerhard Lakemeyer**[2]
[1]Bonn-Rhein-Sieg University of Applied Science, Sankt Augustin, Germany
e-mail: {anastassia.kuestenmacher,paul.ploeger}@h-brs.de
[2]RWTH Aachen University, Aachen, Germany e-mail: gerhard@ksg.rwth-aachen.de

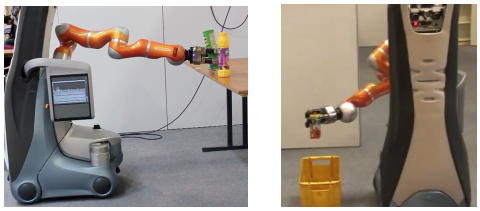
Abstract

Robots interacting in complex and cluttered environments may face unexpected situations referred to as external faults which prohibit the successful completion of their tasks. In order to function in a more robust manner, robots need to recognise these faults and learn how to deal with them in the future. We present a simulation-based technique to avoid external faults occurring during execusion releasing actions of a robot. Our technique utilizes simulation to generate a set of labeled examples which are used by a histogram algorithm to compute a safe region. A safe region consists of a set of releasing states of an object that correspond to successful performances of the action. This technique also suggests a general solution to avoid the occurrence of external faults for not only the current, observable object but also for any other object of the same shape but different size.

1 Introduction

Service robots have to perform common household tasks effectively in dynamic and partially known environments. To be truly robust, such robots must to be able to accomplish their actions in a satisfactory manner without supervision. Their performance can be degraded either by internal hardware and software faults or by unexpected *external faults*. Such external faults can occur instantaneously, are sporadic in nature and are largely unforeseeable at the design stage. As it is impossible to fully supply the robots with knowledge of all possible unexpected situations a priori, the robots must determine the causes of *external faults* on their own to avoid their occurrence in the future.

Figure 1 illustrates two typical external fault scenarios. In scenario 1, a service robot Care-O-Bot III [1], has to place an object (the green crisp can) on a table. If the robot were to release the can in its current position it would likely fall over. In scenario 2, the Care-O-Bot III successfully drops an object into a box. However, the robot may fail if the box were already full of other objects. Our following presents a technique that increases



(a) Scenario 1 (b) Scenario 2

Figure 1: Examples of external faults

the ability of service robots to overcome *external faults*. This technique focuses on the actions of manipulator involved in the release of an object. It suggests the safest state space for the release of not only a particular object, but also for any scaled object of the same shape. Releasing the object in a state from this safest state space avoids the occurrence of external faults. To do so we collect three training sets from experiments. One set of samples consists of experiments with a given ob-

[1]http://www.care-o-bot-research.org

ject. The next set contains experiments with an object of the same shape but scaled down to the smallest possible size which the robot is able to grasp. The third set contains experiments with the largest graspable object of the same shape. The algorithm's task is to maintain a good estimation, namely state space, of the positive outcomes (successful execution of the releasing action) within each of the three training sets and to find the common intersection of these three state spaces. The algorithm assumes that by monitoring the post-conditions of an executed action external faults are detected. We also assume availability of a simulation that shows an example of the expected behaviour of the manipulated object for the case of successful completion of the executed action.

2 Related Work

Methods for detection, identification and handling of faults are frequently discussed in robotics literature. In his survey on execution monitoring Pettersson [7] groups the existing approaches into three classes: analytical, data-driven and knowledge-based. In robotics the terms fault diagnosis and fault detection and isolation are used as synonyms for execution monitoring [7]. Fault diagnosis techniques deal mostly with internal faults. These methods focus on model-based monitors, where the models of possible faults are available a priori. Although it is impossible to describe all interaction faults in dynamic environments, the model-based techniques allow the detection of unexpected situations by comparing the nominal/desired behaviour of a robot with its observations. Pettersson *et al.* and Mendoza *et al.* [6] [5] present work, in which external faults are detected from the robot's behaviour, rather than from a predictive model. We present an alternative technique which utilizes simulation in order to find a general solution for external faults during releasing actions. The inspiration for using simulation to handle external faults comes from the PhD work of Zickler [8] PhD in which a physics engine is used as a black-box to compute a robot motion planning in a complex environment. In another study Jorgensen *et al.* [4] use a simulator to address the problem of stable placement of objects onto both plane and complex surfaces. Similar to our approach, the authors use a simulation to produce a set of labeled samples. To compute an optimal drop pose, they suggest two methods. In the first method they fit the largest enclosed ellipsoid to the positive samples. In the second they apply kernel density estimation to estimate probability of each pose. The optimal pose corresponds to the pose with the largest success probability. The methods of Jorgensen *et al.* assume only two relevant parameters. However, for our scenarios (1,2) we need more than two parameters because of dimensional space. Jiang *et al.* [2] propose a learning approach for placing various objects in different places. The authors use a simulator to generate training and test datasets. The created data sets are then labeled manually and used as inputs for supervised learning algorithms to predict the optimal areas for a stable placement. Johnston and Williams [3] present the architecture *Comirit* for common sense reasoning, in which they combine 3D simulation with formal logic. Their framework generalizes the method of analytic tableaux to allow both logical terms and simulation objects within a single search structure. The authors represent each object as an annotated, connected set of vertices, where annotations are used to describe the local physical properties of the object. Such representation requires complex modelling for each single object.

3 Approach

This work is the updated version of our previous *simulation based approach using example simulation* (SBAES)[1]. The previous technique enabled the robot to suggest the safest releasing state of an object for successful performance of its action or to predict the behaviour of an object for a given releasing state.

Previous work

The SBAES approach was presented as a three step scheme, that requires two inputs: 1) an example simulation that showed the desired behaviour of the object for a given action and, 2) a definition of the planning operator of the action that results in a detected external fault. In its first step, the scheme used the example simulation to find a logical description of the expected behaviour of the released object. The description consists of two logical sentences, namely states of the object at the start and at the end of the simulation. In its second step, the approach found the limits of parameters of the object using this description. Each parameter corresponds to a physical property of the manipulated object and the values of these parameters define the releasing state of this object. The approach utilized them to create different examples of the releasing state of the object. With the help of the description vocabulary

and the description of the sample behaviour the approach labelled these examples as either desired or undesired. In step three, these labelled examples were exploited by a learning algorithm which we referred as *N-Bins*, to suggest a safest releasing state. The results of the learning algorithm were used by the SBAES to modify the releasing action of the robot.

The main objective of the current approach is not only to find a safe releasing state of the current manipulated object but also to generate a solution for other objects with the same geometrical shape but variations in scale. This problem can not be solved using our previous approach (as described above), because *N-Bins* algorithm only suggests one optimal releasing state instead of a space of safe states. The other disadvantage of the SBAES approach is that it assumes the parameters are completely independent of each other. This assumption leads to loss connection between the parameters whereas the closer the released object is to the edge of the table, the lower the value of the releasing height should be.

Approach

This approach mainly focuses on the improvement of Step 2 of the SBAES technique, finding safe space for parameters' values to avoid the occurrence of faults. Similar to SBAES the current approach assumes the settings of plan based robotic systems, in which a robot is able to detect the occurrence of a fault at the planning level by monitoring the postconditions of an executed action. It also assumes that the causes of the external faults are limited to natural physical phenomena (there is no external agent involved in the occurrence of faults).

Additionally our technique expects three inputs: 1) a model of the domestic environment, 2) an example simulation of the actual action (the technique only needs the emulation of a given action, the release of any object over any surface) and 3) postconditions to be checked for successful completion of the action. Before applying our algorithm to compute safe regions, we need to perform necessary preparations. First, the given simulation of a domestic environment has to be updated by the later changes in the robot's current environment. For instance, the model of the table in the simulation has to be changed if some other objects are later placed on the table. The next preparation is to get the limits of parameters to generate the training set. In this work we consider six parameters which correspond to location and orientation of the object. To define the initial limits we can use the SBAES approach. The main disadvantage is its strong dependence on a given example simulation which is modelled for a particular released object and a specific surface. But originally a service robot is equipped with knowledge about the sizes of the objects in its environment, own position, dextrous workspace of the manipulator, desirable final state of the object etc. This knowledge can be applied to estimate the limits of each parameter. Later during the simulation these limits are used to randomly select values of the parameters to generate training examples. Each simulated instance is labeled based on given postconditions. We generate three sets of examples for three objects of different size. One set of samples consists of experiments with the current object. The next set includes those experiments with an object of the same shape but scaled down to the smallest possible graspable size. Similarly, the third set contains experiments with the largest graspable object of the same shape. To ensure the ability of the manipulator to grasp of an object we use a bounding box as rough estimation of its size. Below we introduce a notation and illustrate details of the technique.

Assume that for a given action we generated m training instances. These instances are stored as a $m \times (n+2)$ matrix $AllSamples$. $AllSamples$=$\{(State^i, Label^i, ExpSize^i), i = 1, 2, \ldots, m\}$, where $State^i \in \mathbb{R}^n$ are input values of the parameters of an object (i.e. its releasing state) and n is the total number of parameters (in our example they correspond to $x, y, z, \rho, \theta, \phi$, where ρ, θ, ϕ are respectively $roll, pitch, yaw$). $Label^i \in \{-1, 1\}$ are *negative* and *positive* outputs of experiments. $ExpSize^i \in \{small, midium, large\}$ are different categories of experiments which correspond to the experiments with objects of the actual, smallest and largest graspable sizes. $threshold$ is a scalar which shows the desired probability of a positive outcome of the experiment for the safe state space of the parameters. Algorithm 1 shows the pseudocode for finding the common safe state space for given $AllSamples$ and $threshold$. The FIND-SAFE-STATE-SPACE algorithm starts with computing the safe regions SR_{size} for the training set of each experiment category $size \in \{small, middle, large\}$. Then in line '9' it calculates the intersection of these safe regions. This intersection consists of the limits of the parameters where the current action can be performed

Algorithm 1 FIND-SAFE-STATE-SPACE

Input: $AllSamples, threshold$
Output: SR

1: **for all** $size \in \{small, middle, large\}$ **do**
2: $AllSamples_{size} \leftarrow$ Select instances according to experiment category $size$
3: ($PosSamples_{size}$,$NegSamples_{size}$) $\leftarrow$ Split instances according to their labels
4: $Weights \leftarrow$ WEIGHT-INDEXES ($PosSamples_{size}$, $NegSamples_{size}$)
5: $HyperGrid \leftarrow$ Partition the space of $AllSamples_{size}$
6: F_{size},$\{HR_{size}\}\leftarrow$FIND-HIST($Weights$, $PosSamples_{size}$,$HyperGrid$,$threshold$)
7: $SR_{size} \leftarrow$ Extract HR: $F > threshold$
8: **end for**
9: SR=$SR_{small} \bigcap SR_{middle} \bigcap SR_{large}$

safely.

To find SR_{size} the FIND-SAFE-STATE-SPACE algorithm splits the corresponding training set $AllSamples_{size}$ into two matrices $PosSamples_{size}$ and $NegSamples_{size}$ based on the labels of the samples, where $PosSamples_{size}$ contains only the positive instances and $NegSamples_{size}$ contains only the negative instances.

These matrices are used by the WEIGHT-INDEXES algorithm to calculate the importance of each parameter in the behaviour of the objects. This algorithm is based on the procedure described in [1], p. 36. The main difference is that WEIGHT-INDEXES uses the Kolmogorov-Smirnov test (KS-test) to exclude parameters which are not important for the outcome of an experiment. (line '3' in Algorithm 2). The remaining weight indexes are calculated using the measures of the first four statistical moments (i.e. mean μ_p, variance σ_p, skewness $skew_p$ and kurtosis $kurt_p$) of the distributions of the values of each parameter. The differences ($\Delta\mu_p$, $\Delta\sigma_p$, $\Delta skew_p$ and $\Delta kurt_p$) between statistic moments of corresponding columns of $PosSamples$ and $NegSamples$ are used in calculating the so-called importance IMP_p of each parameter. IMP_p is further utilized for computing the weight of the parameter (see line '2' Algorithm 2). For any parameter, higher value of W_p shows that the final state of the object in the simulation is more sensitive to the value of that parameter.

We associate the safe regions SR_{size} of each experiment category $AllSamples_{size}$ with the intervals on the parameter space $(x, y, z, \rho, \theta, \phi)$, where the amount of positively labeled instances reaches the given $threshold$. To find such regions we decompose the given parameter space into hyper-rectangles and count the number of positive instances that fall into each of the hyper-rectangles. A hyper-rectangle is defined as the Cartesian product of partitions. A well-known way to build a set of hyper-rectangles with corresponding counts is a multidimensional histogram.

Algorithm 2 WEIGHT-INDEXES

Input: $PosSamples$, $NegSamples$
Output: $Weights \leftarrow$ A vector composed of W_p for each parameter

1: **for** each parameter, $p \in 1, 2, \ldots, n$ **do**
2: $W_p = \frac{IMP_p}{\sum_{i=1}^{n} IMP_i}$ where:
3: $IMP_p \leftarrow 0$ **if** KS-test($PosSamples_p$)>0
4: **else** $IMP_p \leftarrow |\Delta\mu_p| + |\Delta\sigma_p| + |\Delta skew_p| + |\Delta kurt_p|$
5: **end for**

The FIND-HIST function shown in Algorithm 3 presents a pseudocode to compute multidimensional histogram where the counts of positive samples occurring in certain ranges of parameters's values are equal or higher than the $threshold$. The weight vector $Weights$, the matrix $PosSamples$ (positive instances of the one of the experiment's category e.g. $small$, $middle$ or $large$), the partition of the space of $AllSamples$ of the current category in hyper-rectangles $HyperGrid$ and $threshold$ are the input arguments to the algorithm. In this algorithm the multidimensional histogram is specified as a set of hyper-rectangles $\{HR^j, j = 1, \ldots, h\}$, and multi-dimensional array F showing how many instances fit within a certain hyper-rectangle.

The FIND-HIST algorithm finds the index/-es of the parameters corresponding to the first largest weight values from the weight vector $Weights$ and returns the $MostImportantDims$ vector in line '3'. Then it selects the $CurrentData$ sub-matrix of the $PosSamples$ matrix consisting of the $MostImportantDims$ columns and the $CurrentDim$ sub-set of the $HyperGrid$. $CurrentDim$ consists of partitions (bins) which correspond to the $MostImportantDims$ dimensions. Using $CurrentDim$ and $CurrentData$ the Algorithm 3 in line '6' computes the counts F and the hyper-rectangles $\{HR\}$. If the termination criteria in line '2' is satisfied, the current F and $\{HR\}$ are the solutions. Otherwise

the algorithm extends the $MostImportantDims$ vector with the index of next parameter according to the next highest weight, constructs a new $CurrentData$ matrix and $CurrentDim$ array and computes a new F and $\{HR\}$.

The output of the FIND-HIST algorithm (F, $\{HR\}$) is used in FIND-SAFE-STATE-SPACE to extract the safe region SR_{size} for the current training set. By repeating the described procedure for the training set of every experiment, we get a collection of safe regions $\{SR_{small}$, SR_{middle}, $SR_{large}\}$. In general, the intersection of these safe regions SR=$SR_{small} \bigcap SR_{middle} \bigcap SR_{large}$ is the particular solution for computing limits of the parameters in which the current action can be performed safely with the desired probability for any graspable object of the given shape.

Algorithm 3 FIND-HIST

Input: $Weights, PosSamples, HyperGrid, threshold$

Output: F - counts on relevant dimensions in total parameter space, HR - Cartesian product of partitions

1: $dimF \leftarrow 1, maxCount \leftarrow 0$
2: **while** $maxCount \leqslant threshold$ **and** $dimF \leqslant$ number of non-zero entries of $Weights$ **do**
3: $MostImportantDims \leftarrow$ Indexes of the first $dimF$-th elements with the largest weights from $Weights$ array
4: $CurrentData \leftarrow PosSamples$ projected to $MostImportantDims$
5: $CurrentDim \leftarrow HyperGrid$ projected to $MostImportantDims$
6: F, $\{HR\} \leftarrow$ build histogram for $CurrentData, CurrentDim$
7: $maxCount \leftarrow$ maximal value of F
8: Increment $dimF$ by 1
9: **end while**

The main challenge in the construction of a histogram is the selection of $HyperGrid$, that is an optimal number of bins or intervals for hyper-rectangles. According to the FIND-HIST algorithm, it first computes a histogram for one dimension (one parameter). If the resulting histogram does not satisfy termination conditions, the algorithm gradually increases the number of dimensions until the desired histogram is found. Hence the number of dimensions vary from 1 to the total number of non-zero entries of the $Weights$ vector. On the other hand the probability distribution of a given training set based on a random sample is uniform. The histogram constructed from them should have in each bin/hyper-rectangle about the same number of elements in average. By experimentation we show that for a successful performance the histogram of the highest dimension, which is the number of all parameters, should have a minimum 50 instances in average. This value is used to compute the number of bins for each parameter.

4 Results

In this section we report the results of applying the described algorithm on three experiments in which different objects were released over other objects or over an open container. The releasing states of different objects are calculated using $47{\cdot}10^6$ instances for each experiment category[2].

Experiment 1 (release a die on a table):

The example simulation shows a die being dropped on a table. From its knowledge the robot extracts the specifications of these objects[3]. Using these specifications we create three training sets for the releasing action. Each instance of these sets is labelled using postconditions' constrains[4].

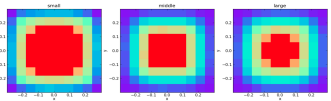


Figure 2: Safe regions of the die for experiment 1

Figure 2 shows the estimated safe regions (red areas) for each experiment category. WEIGHT-INDEX algorithm estimates the following importance order of the parameters: $x, y, z, pitch, yaw, roll$, where the weight of $roll$=0 because according to KS-test $roll$ does not have an impact on the outcome of the experiment ($roll, pitch, yaw$ correspond to ρ, θ, ϕ). The termination conditions are satisfied for the two

[2]With the help of parallelization on 48-core PC (4 × AMD Opteron™6174 12-core 2.2 GHz) the average time per instance is less then 0.8ms

[3]Table's dimensions: 0.47m(h)×0.6m(l)×0.6m(w), die's dimensions: small 0.005m×0.005m×0.005m, middle 0.01m×0.01m×0.01m, large 0.015m×0.015m×0.015m

[4]The experiment performed successfully if the die lies on the table: $Xdie_{final}$∈[-0.3,0.3], $Ydie_{final}$∈[-0.3,0.3] and $Zdie_{final} \approx 0.74$

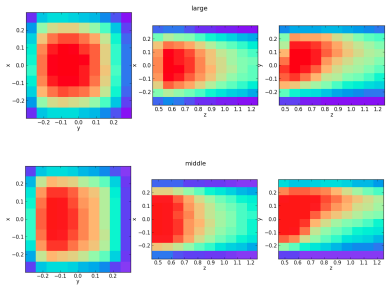


Figure 3: Safe regions of the die for experiment 2

dimensional histogram built for x and y parameters. The intersection of these safe regions is the safe region for releasing the die of the largest size.

Experiment 2 (release a die on an inclined table):
The experiment set up is similar to experiment 1, the only difference is the top of the table tilted at 45 degree. The safe region for the *small* die consists of two (x, z) parameters, but the safe regions for the *middle* and *large* objects include three dimensions (x, y, z), see Figure 3.

Experiment 3 (release a pencil into a box):
The example simulation shows a pencil being placed into an empty container. If the initial inputs for generating the training set are spread too far like in the previous experiments, where $\rho, \theta, \phi \in [-\pi, \pi]$, the algorithm requires more instances to cover all possible behaviours of the object. Hence, it may be necessary to refine them using additional techniques. For example by knowing the object's symmetry groups we can utilize the Euclidian transformation to reduce the limits of ρ, θ, ϕ to $[0, \frac{\pi}{2}]$ interval. Taking into consideration the information about the final orientation of the object (it should stand in the container) we can restrict ρ, θ even more. With the new parameters the algorithm is able to find six dimension safe space.

5 Conclusions and Future Work

The main contribution of this work is to enable a robot using an example of its action to learn how to execute this action in the safest way. We also show that a robot estimates safe performance of an action not only for the given object but also for any graspable object of the same shape in general. There still remain several issues for improving the generalization process, e.g. extending the given algorithm to different objects of similar shape categories. It is obvious that objects from the same category often share similar placing patterns. For example, two cylindrical objects (e.g. a crisp can and a big ketchup bottle) will probably have similar safe regions. It is important to consider a stable release with respect to other objects as well. The object should be placed in its preferred location and orientation in such a way that the other objects on the table will stay at the same positions. If the robot learns the safest placement once, it can still be able to perform this action successfully if the number of objects or their positions on the table are changed. In our algorithm we select a number of bins to compute a histogram empirically, but a more efficient way of such a selection is required. The next step is to extend this work for other robotic actions.

References

[1] N. Akhtar. Improving reliability of mobile manipulators against unknown external faults. Technical report, BRS University of Applied Sciences, Sankt Augustin, Germany, 2012.

[2] Y. Jiang, M. Lim, C. Zheng, and A. Saxena. Learning to place new objects in a scene. *I. J. Robotic Res.*, 31(9):1021–1043, 2012.

[3] B. Johnston and M. Williams. Comirit: Commonsense reasoning by integrating simulation and logic. In *Artificial General Intelligence*, pages 200–211, 2008.

[4] J. A. Jorgensen, L. Ellekilde, and H. G. Petersen. Handling uncertainties in object placement using drop regions. *Proceedings of ROBOTIK*, pages 1 –6, 2012.

[5] J.P. Mendoza, M. Veloso, and R. Simmons. Motion interference detection in mobile robots. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[6] O. Pettersson, L. Karlsson, and A. Saffiotti. Model-free execution monitoring in behavior-based robotics. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 37(4):890–901, 2007.

[7] O. Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53:73–88, 2005.

[8] S. Zickler. *Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments*. PhD thesis, Carnegie Mellon University, May 2010.

A Study of Diagnosability in Dynamic Systems: Integral and Derivative Causality

Hamed Khorasgani[1] and **Gautam Biswas** [2]

[1]Vanderbilt University and Institute for Software Integrated Systems, Nashville, TN, USA
email: hamed.g.khorasgani@vanderbilt.edu
[2] Vanderbilt University and Institute for Software Integrated Systems, Nashville, TN, USA
email: gautam.biswas@vanderbilt.edu

ABSTRACT

This paper attempts a systematic analysis of diagnosability when comparing derivative versus integral causal forms of dynamic system models. We formally define the notions of structural detectability and structural isolability, and use these definitions in a case study where we compare these factors for a three-tank system model. We prove that integral and derivative causality can lead to different isolability properties and by using derivate and integral causality forms of the models concurrently we can maximize fault isolability of the system.

1 Introduction

The control systems-based FDI community and AI-based DX community have developed a number of methodologies for fault detection and isolation in dynamic systems using structural models [1-6]. Structural methods for diagnosis have a number of advantages. These methods are based on the interconnectedness of system components and variables, and numerical values of the parameters do not affect the design of the diagnostic process. Therefore, structural methods are more general and easier to implement, and they can be easily applied to different systems and domains. Also, many common complexities in other methods like existence of answer or singularity of the matrices in the design process do not directly affect the structural methods.

The DX community has developed several structural approaches for diagnosis, e.g., the Temporal Causal Graph (TCG) approach which derives Qualitative Fault Signatures (QFS) [2] and the Possible Conflicts (PC) approach that exploits local redundancy in system measurements [3]. In parallel, the FDI community has developed ARR schemes (e.g., [4]) based on system equations. One of these is the Diagnostic Bond Graph approach that derives ARRs in an organized manner from system bond graph models [1]. Since system equations can be reformulated in many different ways to exploit the analytic redundancy relations for fault detection and isolation, researchers have often restricted the form of the equations used to integral or derivative causality forms. The integral causality form of the models has advantages that they are more robust to measurement noise, but they have the disadvantage that the initial system state has to be known for residual generation. The derivative causality models do not require the initial state, but computing derivatives in noisy environments is hard, and may lead to false alarms, and incorrect diagnoses.

The diagnosis community has often debated if the method of representing the model of a dynamic system affects the inferred diagnosability of the system. On the surface this may not appear to be the case, because irrespective of how the constraints are represented, the collective set of equations defines the dynamic behavior of the system. In this paper, we undertake a preliminary investigation of this topic, by comparing the diagnosability of system models represented in integral and derivative causality – the two most common forms of representation for dynamic system models.

Recently, Frisk, et al. [7] presented through an example that using different forms of causality in the system equations can lead to different isolability properties. The results of this paper is a primary motivation for this paper, i.e., study in more detail how different forms of causality can lead to different isolability properties and how we can utilize the information provided by integral and derivative causality models concurrently to maximize isolability in the system. In this paper, we make an attempt to make these results more systematic.

The rest of this paper is organized as follows. In section 2, we formally present the notions of integral and derivative causality forms of system behavior models. The three tank system is used as a case study to demonstrate the various concepts and the results derived in this paper. In section 3, we first formally define the no-

tions of detectability and isolability in dynamic systems. Then Possible Conflicts (PC), a structural diagnosis approach developed by Pulido and Alonso [3] is utilized to demonstrate the diagnosability results on the three tank system using integral and derivative causality. In previous work [10], we have demonstrated the equivalence between the PC and ARR approaches to diagnosis.

Section 5 discusses an approach where the derivative and integral causality forms of the dynamic system models are integrated to achieve the highest levels of diagnosability in a system, for a given set of sensors. Finally, Section 6 discusses the advantages and drawbacks of using the two forms of causality concurrently, and concludes by laying out a brief plan for future work.

2 Test Case: A Three Tank System

To study fault detectability and isolability in dynamic systems a simple three tank system model shown in Figure 1 is considered as a running example for this paper.

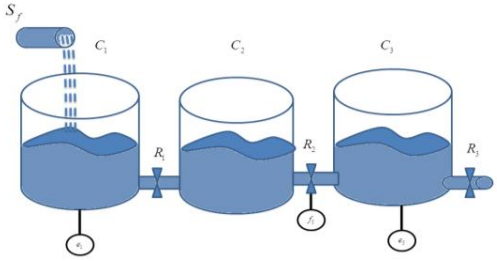


Figure 1: Three tank system configuration.

The three tank system consists of a flow input source (S_f), three tanks (C_1, C_2 and C_3) and three valves (R_1, R_2 and R_3). Also three sensors measure the pressure of the first tank (e_1), the volumetric flow rate of the second valve (f_5) and the pressure of the last tank (e_3). Six possible faults in the system components (R_1, R_2, R_3, C_1, C_2 and C_3) are considered in this study.

This section first reiviews Bond Graph modeling language and its relationship with structral fault diagnostability and then uses bond graph model to derive system equations in integral and derivative causalities.

2.1 Bond Graph Modeling Methodoloy

Bond Graphs are topological, energy based, modeling methodology for physical processes [8]. Nodes in the bond graphs represent elements of the dynamic system and directional links or bonds show energy path and its positive direction between the elements. Effort and flow are the energy variables in the bond graph language. They represent different variables in different domains. For instance in the hydraulic domain effort represents pressure and flow is volumetric flow rate. System elements in the bond graphs are modeled as sources (source of effort and source of flow), energy storage elements (capacities and inertias) and dissipative elements (resistors). Two ideal junctions (0– and 1–junctions) are also defined in the bond graph modeling language.

Series (1–)junctions are common flow junctions and parallel (0–)junctions are common effort junctions. The bond graph model of the three tank system in Figure 1 is shown in Figure 2. There is an effort and a flow variable associated with each bond (half arrow) in the bond graph. The product of the effort with flow variable represents the rate of energy flow between connected components. In Figure 2 the bonds connected to a common effort junction (0–junction) only one effort variable and for the bonds connected to a common flow junction (1–junction) only one flow variable is independent, and determines the value of that variable on all incident bonds.

Mosterman and Biswas [2] define the set of hypothesized faults candidates for a system as the parameters of components of its bond graph model. In [5] bond graphs are used to derive ARRs and [9] suggests a method to derive PCs from the bond graph model of the system. Based on these similarities we utilize bond graph models as a common framework for comparing diagnosability of different structural methods [10]. In bond graphs, causality is represented by a vertical bar at the side of link, and this defines the side of the effort receiver. For example, in Figure 2 R_1 represents a resistor with input effort, and, consequently, the flow value is the output defined by the equation associated with the resistor, $f_3 = \frac{e_2}{R_1}$. There are four kinds of causalities in the bond graphs. (1)-Fixed causality :for sources and nonlinear elements which can accept one form of causal model. (2) Constrained causalities for junctions: 0–junctions can accept effort only from one of the connected links and similarly, 1–junctions can accept flow only from one of the connected links. (3) Preferred causality: for energy storage elements. If we prefer integral causality, we assign the causality in a way that capacities receive flow and inertias receive effort. For derivative preferred causality, capacities receive effort and inertias receive flow. (4) Arbitrary causality for dissipative elements: Assigning a specific causality form to the bond graph represents the computational order that the model should be solved. For example, in figure 2 the assigned causality to R_1 informs

the simulator to compute e_2, and then use e_2 to compute f_3. In the next two subsections the equations of the three tank system model are derived in integral and derivative causality forms.

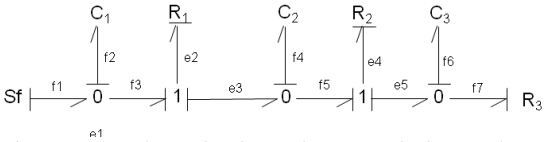


Figure 2 Bond graph of 3 tank system in integral causality.

2.2 Deriving System Equations: Integral versus Derivative Causality in Dynamic Systems

To derive the equations of the system in integral causality form we assign integral causality to the bond graph bonds as shown in Figure 2. Then for each element and junction we write the associted equation based on the assigned cauasality, as shown below:

$$
\begin{aligned}
&f_2 = f_1 - f_3 && e_5 = \frac{1}{C_3}\int f_6 \\
&e_1 = \frac{1}{C_1}\int f_2 && f_7 = \frac{e_5}{R_3} \\
&e_2 = e_1 - e_3 && e_4 = e_3 - e_5 \\
&f_3 = \frac{e_2}{R_1} && f_5 = \frac{e_4}{R_2} \\
&f_4 = f_3 - f_5 && f_6 = f_5 - f_7. \\
&e_3 = \frac{1}{C_2}\int f_4
\end{aligned}
\qquad (1)
$$

For modeling dynamic systems usually integral causality is prefered for several reasons. The most important one is that derivative operator is not causal, i.e., to calculate the derivation of a variable at time t its value at the next sample time is needed.

$$\frac{de(t)}{dt} = \lim_{\Delta t \to 0} \frac{e(t+\Delta t) - e(t)}{\Delta t}. \qquad (2)$$

However, the simulator can wait for one sample time and calculate the derivative. So it is possible to simulate three tank systems in derivative causality as well. To derive the equations of the system in derivative causality we assign derivative causality to the bond graph model of the system. Figure 3 shows the bond graph model of the three tank system in derivative causality. Set of equations of the elements and junctions of the bond graph of three tank system in derivative causality, form the set of system equations in derivative causality as represented in (3).

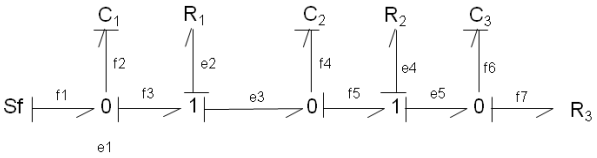


Figure 3 Bond graph of three tank system in derivative causality.

$$
\begin{aligned}
&f_3 = f_1 - f_2 && e_3 = e_4 + e_5 \\
&f_2 = C_1 \frac{de_1}{dt} && e_4 = R_2 f_5 \\
&e_1 = e_2 + e_3 && f_7 = f_5 - f_6 \\
&e_2 = R_1 f_3 && f_6 = C_3 \frac{de_5}{dt} \\
&f_5 = f_3 - f_4 && e_5 = R_3 f_7. \\
&f_4 = C_2 \frac{de_3}{dt}
\end{aligned}
\qquad (3)
$$

To complete the set of system equations input and outputs variables are expressed as:

$$u = f_1 \qquad (4)$$

$$y_1 = e_1, \quad y_2 = f_5, \quad y_3 = e_5. \qquad (5)$$

Combinations of integral and derivative causality, i.e., mixed causality is not considered in this paper. In the next section we will see how integral and derivative causalities can lead to different fault isolation properties.

3 Fault Detection and Isolation: Possible Conflicts Approach

Possible conflicts (PCs) are localized methods for structural diagnosis. Given a set of system equations, PCs capture all the minimal subsets of constrains which produce inconsistencies when faults occur. Any constraint in the system which contains fault components and has sufficient analytical redundancy to capture faults is a possible conflict. In the first part of this section, detectable and isolable faults are defined. Then possible conflicts method is used in integral and derivative causality forms to isolate the faulty components in the three tank system.

3.1 Detectability and Isolability in Dynamic Systems

In this section, we start with relevant definitions.

Definition 1: Given the set of component faults $F = \{f_1, ..., f_n\}$ and the set of possible conflicts

$P = \{p_1, ..., p_m\}$, the fault connection matrix (FCM) is a $m * n$ matrix and its elements are defined as:

$$FCM(i, j) = \begin{cases} 0 & \text{if } p_j \text{ does not include } f_i \text{ component} \\ 1 & \text{if } p_j \text{ includes } f_i \text{ component.} \end{cases}$$

Using definiton 1 we can define detectable and isolatable faults [10].

Definition 2: A fault parameter f_i is structurally detectable, if there exists a non-zero entry in the FCM for at least one possible conflict.

Definition 3: Fault parameter f_i and f_j are structurally isolable from each other if they have different signatures in the FCM.

Using these definitions the isolability of different component faults for integral and derivative causality is analyzed in the following subsections.

3.2 Diagnosability of System Faults using PC in Integral Causality

Fault detection and isolation starts with the system dynamic model. In this subsection, the set of system equation in integral causality (1) is applied to derive Possible Conflicts. Consider $\frac{1}{D}$ as the integral operator we have:

$$e_2 = e_1 - e_3 = e_1 - \frac{f_4}{C_2 D} = e_1 - \frac{\frac{e_2}{R_1} - f_5}{C_2 D} \quad (6)$$

$$\Rightarrow e_2 = \frac{R_1 C_2 D}{(R_1 C_2 D + 1)}(e_1 + \frac{f_5}{C_2 D})$$

and $$e_1 = \frac{1}{DC_1} f_2 = \frac{1}{DC_1}(f_1 - f_3) = \frac{1}{DC_1}(f_1 - \frac{e_2}{R_1}). \quad (7)$$

By substituting (6) in (7) we get:

$$e_1 = \frac{1}{DC_1}(f_1 - \frac{R_1 C_2 D}{(R_1 C_2 D + 1)R_1}(e_1 + \frac{f_5}{C_2 D})). \quad (8)$$

Since e_1, f_1 and f_5 are all known, equation (8) is a possible conflict. From (1) one can also conclude that:

$$e_3 = \frac{f_4}{DC_2} = \frac{f_3 - f_5}{DC_2} = \frac{\frac{e_2}{R_1} - f_5}{DC_2} = \frac{e_2 - R_1 f_5}{DC_2 R_1} = \frac{e_1 - e_3 - R_1 f_5}{DC_2 R_1}$$

$$\Rightarrow e_3 = \frac{e_1 - R_1 f_5}{(1 + DC_2 R_1)}, \quad (9)$$

and:

$$f_5 = \frac{e_4}{R_2} = \frac{e_3 - e_5}{R_2}. \quad (10)$$

By substituting (9) in (10) one can get:

$$f_5 = \frac{e_1 - R_1 f_5 - e_5(1 + DC_2 R_1)}{R_2(1 + DC_2 R_1)}. \quad (11)$$

Since e_1, e_5 and f_5 are all known (11) is also a possible conflict. For e_5 we can say:

$$e_5 = \frac{f}{DC_{3\,6}} = \frac{f_5 - f_7}{DC_3} = \frac{f_5 - \frac{e_5}{R_3}}{DC_3} = \frac{R_3 f_5 - e_5}{DC_3 R_3}. \quad (12)$$

Here also e_5 and f_5 are both known so equation (12) is also a conflict. Based on (8), (11) and (12) we can derive FCM as:

Table 1: FCM in Integral Causality

	R_1	R_2	R_3	C_1	C_2	C_3
e_1	*1*	*0*	*0*	*1*	*1*	*0*
f_5	*1*	*1*	*0*	*0*	*1*	*0*
e_5	*0*	*0*	*1*	*0*	*0*	*1*

Using FCM, one can derive the maximum isolability matrix as:

Table 2: Maximum Isolability Matrix in Integral Causality

	R_1	R_2	R_3	C_1	C_2	C_3
R_1	×				×	
R_2		×				
R_3			×			×
C_1				×		
C_2	×				×	
C_3			×			×

Table 2 shows R_1 is not isolatable from C_2 and R_3 is not isolatable from C_3 but all the other faults are isolatable from each other.

3.3 Diagnosability of System Faults using PC in Derivative Causality

To perform fault detection and isolation in derivative causality we start with the set of equations in derivative causality (3) and consider the same set of inputs (4) and sensors (5) as the integral causality model. Representing the derivation operator as D and doing some algebraic manipulations we have:

$$e_1 = e_2 + e_3 = R_1 f_3 + e_4 + e_5 = R_1(f_1 - C_1 D e_1) + R_2 f_5 + e_5. \quad (13)$$

Since e_1, f_5 and e_5 are all known so equation (13) is a possible conflict. We also have:

$$f_5 = f_3 - f_4 = f_1 - f_2 - C_2 D e_3 = f_1 - C_1 D e_1 - C_2 D(e_4 + e_5)$$
$$= f_1 - C_1 D e_1 - C_2 D(R_2 f_5 + e_5), \quad (14)$$

where e_1, f_1, f_5 and e_5 are all known variables and (14) is also a possible conflict. Finally, we can say:

$$e_5 = R_3 f_7 = R_3(f_5 - f_6) = R_3(f_5 - C_3 D e_5). \quad (15)$$

It is clear that (15) is a possible conflict as well. Table 3 shows the FCM for the three-tank system in derivative causality form. The maximum isolability matrix for derivative causality is presented in Table 4.

Table 3: FCM in Derivative Causality

	R_1	R_2	R_3	C_1	C_2	C_3
e_1	*1*	*1*	*0*	*1*	*0*	*0*
f_5	*0*	*1*	*0*	*1*	*1*	*0*
e_5	*0*	*0*	*1*	*0*	*0*	*1*

Table 4: Maximum Isolability Matrix in Derivative Causality

	R_1	R_2	R_3	C_1	C_2	C_3
R_1	×					
R_2		×		×		
R_3			×			×
C_1		×		×		
C_2					×	
C_3			×			×

In this case, R_1 is isolatable from C_2 but R_2 is not isolatable from C_1 and R_3 and C_3 are still non isolatable. The example is quite interesting because it shows changing the causality can change the isolability of the faults. A method to isolate maximum possible faults from each other is suggested in the next section.

4 Concurrent Causality

In this section, we use the definitions presented in section 3.1 and the FCM derived in sections 3.2 and 3.3 to show if we consider derivative and integral causality simultaneously, we can improve the isolability of the faults. Then we use this theoretical discussion to suggest fault isolation using concurrent causality.

4.1 Fault Isolability in Dynamic Systems

Isolability information provided in integral and derivative causalities is discussed in the following theorem.

Theorem: Structural isolability information provided by considering derivative and integral causalities together (concurrent causality) provides equal or more isolability than by considering just derivative or just integral causality.

Proof: To prove this theorem we first prove that the isolability information provided in derivative and integral causality are not the same. To prove this part we simply use contradiction. Assume that isolability information provided by derivative and integral causality are the same. If we can provide one example where integral and derivative causalities lead to different solutions, the first part is proved. Consider the three-tank system in figure 1 with the set of measurements provided in (5). One can see from Table 2 that in integral causality faults in R_1 and C_2 are not isolatable from each other. But Table 4 shows these faults are isolable in derivative causality. Also from Table 4 it can be seen that faults in R_2 and C_1 are not isolable from each other in derivative causality but Table 2 shows they are isolable in integral causality. So it is proved that derivative and integral causality provide different isolability information and the information provided by each of them is not a subset of the other. Therefore, by using both of them we can get more or in some possible cases equal information of using one of them.

4.2 Fault Diagnostic in the Three-tank System using Concurrent Causality

Based on the theoretical discussions in the previous subsection we expect that by using concurrent causality we achieve to the maximum isolability in our case study. Figure 4 shows fault diagnostic structure using concurrent causality.

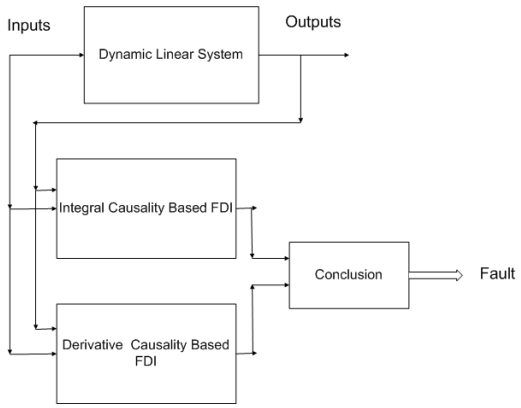


Figure 4 Concurrent causalities.

We know that non of the integral and derivate causality does not provide maximum isolability information. So to isolate maximum faults concurrent causality is applied. To this end equations (8), (11), (12) and equations (13), (14), (15) are all used to derive possible conflicts. In this case FCM is:

Table 8: FCM in Concurrent Causality

	R_1	R_2	R_3	C_1	C_2	C_3
$e_1(i)$	*1*	*0*	*0*	*1*	*1*	*0*
$f_5(i)$	*1*	*1*	*0*	*0*	*1*	*0*
$e_5(i)$	*0*	*0*	*1*	*0*	*0*	*1*
$e_1(d)$	*1*	*1*	*0*	*1*	*0*	*0*
$f_5(d)$	*0*	*1*	*0*	*1*	*1*	*0*
$e_5(d)$	*0*	*0*	*1*	*0*	*0*	*1*

where index i represents integral and index d represents derivative causality. The maximum isolability matrix in this case would be:

Table 10: Maximum Isolability Matrix in Concurrent Causality

	R_1	R_2	R_3	C_1	C_2	C_3
R_1	×					
R_2		×				
R_3			×			×
C_1				×		
C_2					×	
C_3			×			×

It can be seen from Table 10 that the faults C_1 and R_2 or R_1 and C_2 are structurally isolatable in this method.

5 Conclusions

In this paper, we showed that by considering integral and derivative causality simultaneously we can isolate some faults which were not isolatable in just integral or derivative case. However, the important point is that each of the integral and derivative causalities has some limitations. For example, derivative causality is not proper for noisy environments because possibly we have to get derivation from some measurements and in noisy environments it can lead to huge errors and false alarms. On the other hand, integral causality may lead to unstable residuals. Also considering both of the causalities needs twice computations. So in the cases that we do not have these limitations concurrent causality could be considered as an interesting tool to isolate faults as much as possible. In future work, we will study mixed causality and discus which mixed causality can provide maximum fault isolation in the system.

REFERENCES

[1] Samantaray, A. K., & Bouamama, B. O. (2008). Model-based process supervision: a bond graph approach. Springer.

[2] Mosterman, P. J., & Biswas, G. (1999). Diagnosis of continuous valued systems in transient operating regions. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *29*(6), 554-565.

[3] Pulido, B., & González, C. A. (2004). Possible conflicts: a compilation technique for consistency-based diagnosis. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *34*(5), 2192-2206.

[4] Staroswiecki, M., & Comtet-Varga, G. (2001). Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems. *Automatica*,*37*(5), 687-699.

[5] Samantaray, A. K., Medjaher, K., Ould Bouamama, B., Staroswiecki, M., & Dauphin-Tanguy, G. (2006). Diagnostic bond graphs for online fault detection and isolation. *Simulation Modelling Practice and Theory*, *14*(3), 237-262.

[6] Krysander, M. (2006). Design and analysis of diagnosis systems using structural methods (Doctoral dissertation, Linköping).

[7] Frisk, E., Bregon, A., Aslund, J., Krysander, M., Pulido, B., & Biswas, G. (2012). Diagnosability analysis considering causal interpretations for differential constraints. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *42*(5), 1216-1229.

[8] Karnopp, D. C., Margolis, D. L., & Rosenberg, R. C. (2012). System dynamics: modeling, simulation, and control of mechatronic systems. Wiley.

[9] Bregon, A., Biswas, G., & Pulido, B. (2012). A decomposition method for nonlinear parameter estimation in TRANSCEND. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 42(3), 751-763.

[10] Bregon. A., Biswas, G, Pulido, B, Alonso-Gonzalez, C & Khorasgani, H, (2013) "A Common Framework for Compilation Techniques Applied to Diagnosis of Linear Dynamic Systems" Manuscript submitted for publication.

And Yet Another Variant of Reiter's Complete On-the-fly Hitting Set Algorithm

Ingo Pill and Thomas Quaritsch
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, 8010 Graz, Austria,
{ipill, quaritsch}@ist.tugraz.at

Abstract

Complementing recent research regarding the direct computation of diagnoses in theorem provers, complete hitting set algorithms are still an essential technique in the context of model-based diagnosis. Besides deriving diagnoses from available conflicts, they may even drive the search for those unsatisfiable cores in the problem description. Thus a series of algorithms have been emerging, consecutively aiming to lower required resources. In this paper, we show an extension to Greiner et al.'s variant of Reiter's hitting set algorithm, that like Wotawa's variant aims at minimizing the nodes and corresponding subset-checks needed for maintaining the DAG/tree encoding the search space exploration. First experimental results show the attractiveness of our algorithmic extension.

1 Introduction

With today's complex systems, the identification of root causes for encountered issues is an essential aspect of any development process, regardless of the system domain. Aimed at tackling this issue, the diagnosis community has been proposing a large variety of competing solutions based on different approaches and in a multitude of variants. Minimal hitting set (MHS) algorithms play a central role in several of these approaches, either as a last step "combining" intermediate results in the form of conflicts [de Kleer and Williams, 1987; de Kleer, 2011; Reiter, 1987], or, even driving the search space exploration and the computation of conflicts [Reiter, 1987]. Catching an essential connection between diagnoses and conflicts, they are of interest also when interleaving conflict and diagnosis computation [Stern *et al.*, 2012].

This resulted in a series of corresponding MHS algorithms, where, for instance, Greiner et al.'s HS-DAG [Greiner *et al.*, 1989] is an optimization (and correction) of Reiter's search space exploration idea that he defined in his formalizations regarding consistency-based diagnosis. However, there is still a lot of redundant work in HS-DAG's search, which is reflected, for instance, in the "reuse" of nodes when varying decision sequences lead to the same assumptions. Wotawa aimed with his variant HST [Wotawa, 2001] of Reiter's algorithm at minimizing the subset-checks required for building the tree by implementing a specific structure in the search. In this paper, we show how to implement a somehow related strategy by mimicking to some extent the divide-and-conquer-based strategy present in the Boolean algorithm [Lin and Jiang, 2003] (that repeatedly splits the search space into two parts; those solutions containing some chosen split element e and those not containing e) with the HS-DAG algorithm.

That is, our reasoning about the search space explored in some sub-DAG allows us to actually exclude specific elements from consideration in other sub-DAGs. Exploiting this reasoning we are able to achieve occasionally a significant reduction in the number of nodes derived (and the related subset- and consistency checks). In our first tests, we achieved a runtime speed-up, node reduction, and memory reduction of up to factors 3.77 / 4.15 / 2.72 respectively.

We present our work as follows. In Section 2 we cover the preliminaries, with Section 3 focusing on our extensions to HS-DAG. Following the experimental results reported in Section 4, we draw our conclusions and depict future work in Section 5. Related work is discussed throughout the paper where appropriate.

2 Preliminaries

In the context of model-based diagnosis, Reiter defined in [Reiter, 1987] a minimal hitting set (MHS) algorithm aiming at the computation of diagnoses as the minimal hitting sets of a diagnosis problem's conflicts.

Definition 1. *A* hitting set *for a set CS of sets C_i is a set $h \subseteq \bigcup_{C_i \in CS} C_i$ s.t. $\forall C_i \in CS : h \cap C_i \neq \emptyset$. h is minimal, iff there is no $h' \subset h$ that is a hitting set as well.*

In Reiter's formulations focusing on a system of connected components, he proposed an algorithm for deriving minimal hitting sets for some set *CS* containing the conflicts responsible for encountered inconsistencies between experienced and expected system behavior. His complete algorithm is based on a breadth-first exploration of the search space (exponential in the size of $\bigcup_{C_i \in CS}$) that is organized by maintaining a specific tree. While his algorithm was basically correct, it could miss diagnoses when non-minimal conflicts were used. Greiner et al. presented [Greiner *et al.*, 1989] a corrected version that furthermore uses a DAG instead of a tree. Their algorithm, HS-DAG, is defined for some ordered (i.e., consistently traversable) *CS*, whose C_is may be in arbitrary order or—more favorably—sorted by their cardinality. For our paper to be self-contained, we recap their algorithm in the following, adopting its description in [Greiner *et al.*, 1989]:

1. Let D represent a growing node- and edge-labeled DAG with some initial and unlabeled root node n_0. Proceed with Step 2.
2. Process the unlabeled nodes in D in a breadth-first order. To process a node n:
   (a) Define $h(n)$ to be the set of edge labels on the path in D from root node n_0 to node n ($h(n_0) = \emptyset$).
   (b) Iff for all $C_i \in$ *CS*: $C_i \cap h(n) \neq \emptyset$, then label n with "✓". Otherwise label n with some C_j: C_j is the first set in *CS* s.t. $C_j \cap h(n) = \emptyset$.
   (c) If n is labeled with some $C_i \in$ *CS*, generate for each $c_i \in C_i$ a new edge labeled with c_i. This edge leads to a new node n' with $h(n') = h(n) \cup \{c_i\}$. The new node n' will be processed (labeled and expanded) after all new nodes n_i in the same generation as n (s.t. $|h(n_i)| = |h(n)|$) have been processed.
3. Return the resulting DAG D.

To (correctly) compute only the *minimal* hitting sets of *CS*, Greiner et al. proposed the following pruning enhancements to the algorithm:

1. Reusing nodes: The algorithm will not always generate a new node m as a descendant of node n. There are two cases to consider:
   (a) If there is a node n' in D such that $h(n') = h(n) \cup \{c_i\}$, then let the edge labeled c_i originating from n point to n'. Hence, n' will have more than one parent.
   (b) Otherwise, generate a new node m as destination for the edge labeled c_i as described in the basic algorithm.
2. Closing: If there is a node n' such that $h(n') \subset h(n)$, and which is labeled with "✓", then close node n. Neither a label is computed for n, nor are any successor nodes generated.
3. Pruning: If a priorly unused set C_i is used to label a node, attempt to prune D as described in the following.
   (a) For nodes n' labeled with some $C_j \in$ *CS* such that $C_i \subset C_j$, relabel n' with C_i. Then, for any c_i in $C_j \setminus C_i$, the edge labeled c_i originating from n' is no longer allowed. The node connected by this edge and all of its descendants are removed, except for those nodes with another ancestor that is not being removed. Note that this step may eliminate the node which is currently being processed.
   (b) Interchange the sets C_j and C_i in *CS*. (Note that this has the same effect as eliminating C_j from *CS*.)

Note that the last rule (3) is relevant only if *CS* contains some sets C_i and C_j s.t. $C_i \subset C_j$ and the sets in *CS* are not sorted in respect of their growing cardinality. This is of specific interest when computing *CS* on-the-fly and the theorem prover returns not necessarily minimal conflicts.

In the context of our diagnosis examples in our evaluation, we follow Reiter's definitions [Reiter, 1987], and assume a diagnosis problem to be defined by a set of system components *COMP*, a system behavior description *SD*, and some actual system behavior observations *OBS*. For our examples, *SD* defines the correct behavior of the system in the form of logic sentences $\neg AB(c_i) \Rightarrow \mathit{NominalBehavior}(c_i)$, where assumption predicates $AB(c_i)$ for all $c_i \in$ *COMP* encode whether c_i behaves *abnormally* or not.

Definition 2. *A* conflict C *for (SD, COMP, OBS) is a set* $C \subseteq$ *COMP such that SD* $\cup$ *OBS* $\cup \{\neg AB(c_i) | c_i \in C\}$ *is inconsistent. Iff there is no* $C' \subset C$*, such that* C' *is a conflict,* C *is a* minimal *conflict.*

Definition 3. *A* diagnosis *for (SD, COMP, OBS) is a subset-minimal set* $\Delta \subseteq$ *COMP such that SD* $\cup$ *OBS* $\cup \{\neg AB(c_i) | c_i \in$ *COMP* $\setminus \Delta\}$ *is consistent.*

Proposition 1. *(Theorem 4.4 in [Reiter, 1987])* $\Delta \subseteq$ *COMP is a* diagnosis *for (SD, COMP, OBS) iff* Δ *is a minimal hitting set for the collection CS of conflicts C for (SD, COMP, OBS).*

Consequently, a new label (conflict) for a node n can be computed by a theorem prover on-the-fly as an (ideally subset-minimal) unsatisfiable core in the assumption predicates for a consistency check of *SD* $\cup$ *OBS* $\cup \{\neg AB(c_i) | c_i \in$ *COMP* $\setminus h(n)\}$.

When HS-DAG's search for solutions is limited to some bound k regarding their cardinality, nodes with $|h(n)| = k$ need not be expanded. As a consequence, not all possible conflicts have to be computed to solve such problems.

As we will refer in the discussion of our HS-DAG variant to the Boolean algorithm [Lin and Jiang, 2003], we also rehearse it briefly in order for the paper to be self-contained:

The Boolean approach encodes *CS* as a Boolean formula in disjunctive normal form (DNF), with the individual $C_i \in$ *CS* as conjunction of the corresponding negated bits (propositions) for the components $c_i \in C_i$. To derive all minimal hitting sets for *CS*, a recursive function H containing five rules (considered in ascending order) derives from this *CS* formula another formula encoding the MHSs. That is, the result still needs some subset-checks (or the use of Boolean laws) in order to derive a canonical DNF where the conjuncts represent the individual *MHS*s. Assuming $\mathcal{C}$ a Boolean formula, e an atomic proposition with $\bar{e}$ denoting its negation, and $\bot/\top$ referring to $e \wedge \bar{e}/e \vee \bar{e}$, the function $H(\mathcal{C})$ is defined as:

R1: $H(\bot) = \top, H(\top) = \bot$;

R2: $H(\bar{e}) = e$;

R3: $H(\bar{e} \wedge \mathcal{C}) = e \vee H(\mathcal{C})$;

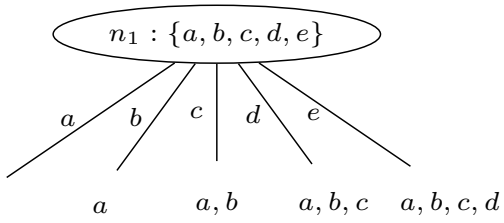
R4: $H(\bar{e} \vee \mathcal{C}) = e \wedge H(\mathcal{C})$;

R5: $H(\mathcal{C}) = e \wedge H(\mathcal{C}_1) \vee H(\mathcal{C}_2)$ for some *arbitrary* atomic proposition e present in $\mathcal{C}$, with
$\mathcal{C}_1 = \{c_i \mid c_i \in \mathcal{C} \wedge \bar{e} \notin c_i\}$ and
$\mathcal{C}_2 = \{c_i \mid \bar{e} \notin c_i \wedge (c_i \in \mathcal{C} \vee c_i \cup \{\bar{e}\} \in \mathcal{C})\}$.

Rule R5 encodes the algorithm's general strategy of how to conquer the search space by splitting the solution space into those solutions not containing split element e (and consequently pruning e from all conjuncts in $\mathcal{C}$) and those solutions including e (removing those conjuncts hit by e from further search). In [Pill and Quaritsch, 2012] we presented a variant where R5 is optimized for a bounded search (and also R4 is slightly improved), which we will use for our performance comparison later on.

3 HS-DAG with Reduced Conflicts

In this section, we show how to reduce redundancies in HS-DAG's search by narrowing the focus for a sub-DAG when it is clear that some solutions would be considered also by



Θ_C for the various sub-DAGs

Figure 1: Conflict reduction concept.

other sub-DAGs. Our basic reasoning is as follows: when expanding a node n (Step 2b), that is creating new edges originating in n, we create for each destination node n' a set $\Theta(n')$ of components for which we will not create new edges in the corresponding sub-DAG. $\Theta(n') = \Theta_C(n') \cup \Theta_N(n')$ for a node $n' \neq n_0$ consists of two subsets. The set $\Theta_C(n')$ of components c_i for which we created outgoing edges from node n already, and $\Theta_N(n') = \Theta(n)$ as the exclusion set inherited from n''s parent n, which we use to propagate our reasoning within a sub-DAG. For the root node n_0 we have $\Theta(n_0) = \Theta_C(n_0) = \Theta_N(n_0) = \emptyset$. In Figure 1 we illustrate the various $\Theta_C(n'')$s when assuming $\Theta(n_1) = \emptyset$ and creating the edges according to their label's appearance in n_1's label C.

Assume now that *CS* contains only subset-minimal conflicts, so that the pruning rule (3) (which we will consider later on) has no effect at all and is thus also not applied. Then we can define the following variant of HS-DAG:

Definition 4. *For our algorithmic variant HS-DAG-RC', besides HS-DAG's standard node-label, assume a further label $\Theta(n) \subseteq$ COMP. Let the algorithm be as the original HS-DAG, but without pruning rule 3 and the following redefinition of Step 2(c):*

If n is labeled with some $C \in$ CS, generate for each $c_i \in C \setminus \Theta(n)$ a new edge labeled with c_i. This edge leads to a new node n' with $h(n') = h(n) \cup \{c_i\}$, where $\Theta(n') = \Theta_C(n') \cup \Theta_N(n')$ with $\Theta_C(n') = \{c_j | c_j \in C$ and we already created an edge labeled c_j from $n\}$ and $\Theta_N(n') = \Theta(n)$. The new node n' will be processed (labeled and expanded) after all new nodes n_i in the same generation as n (s.t. $|h(n_i)| = |h(n)|$) have been processed.

The soundness and completeness of our variant HS-DAG-RC' can be easily seen.

Theorem 1. *For CS containing only subset-minimal sets, the algorithmic variant as of Definition 4 is complete. For any minimal hitting set Δ for CS, D contains some node n labeled ✓ such that $h(n) = \Delta$.*

Proof. (Sketch) Let us assume that there is some minimal hitting set Δ such that there is no node n with $h(n) = \Delta$. Now we know that HS-DAG is complete. As our optimizations so far only concern avoiding the construction of some edges in Step 2(c), these optimizations would have prevented the construction of the necessary node. It is however easily seen, that while we restrict some sequences of element choices that result in a certain combination $h(n) = \Delta$, we do not prevent all possible sequences. That is, for some given Δ and DAG D and starting with the root node n_0, we can always choose the "left-most" branch possible (removing the edge-label c_i from Δ), which has to be an allowed sequence due to the exhaustive nature of step 2(b) and minimizing Θ (i.e., there is no restriction regarding those c_i still left in Δ) along this path. □

Theorem 2. *For CS containing only subset-minimal sets, the algorithmic variant as of Definition 4 is sound. That is, $h(n)$ of any node n in D such that n is labeled with ✓ is indeed a minimal hitting set of CS.*

Proof. (Sketch) Step 2(b) ensures that $h(n)$ for any node labeled ✓ is indeed a hitting set, so that it remains to show that such a $h(n)$ is indeed subset-minimal. Let us now assume that there is some node n labeled ✓ such that $h(n)$ is not minimal. As $h(n)$ is non-minimal, due to Theorem 1, there would have to be some node n' labeled ✓ such that $h(n') \subset h(n)$. For such an n', the closing rule would however have closed n, so that there could not have been such a node n'. Thus for any node n labeled ✓, we indeed have that $h(n)$ is a subset-minimal hitting set for *CS*. □

Now that we have a complete and sound algorithm for subset-minimal conflicts, let us consider the case of non-minimal conflicts. While the respective ideas behind the pruning rule (3) and our reasoning do not interfere, the actual implementation of our reasoning in Step 2(c) does. This comes from the fact that the actually expanded edges interfere with the construction of the sub-DAGs via Θ_C. If an edge then gets pruned, some Θ_C and the Θs in its sub-DAG might need an update. Obviously this also means that some edges which some Θ prevented from being constructed, need to be created with the updated Θs. Our following variant of Definition 4 that includes an adapted *pruning* rule takes care of these issues.

Definition 5. *Let HS-DAG-RC be an algorithm as of Def. 4, but with an updated* pruning *rule 3 from HS-DAG, such that it extends HS-DAG's rule 3(a) as follows:*

Now, for all children n'' of n' update $\Theta_C(n'')$ to $\Theta_C(n'') \setminus (C_j \setminus C_i)$ and for all descendants n''' of some n'' propagate the update (most likely reflected in $\Theta_N(n''')$) accordingly. Then create for all n'' and n''' all the edges that are not avoided anymore (due to the updates to their Θs), and process the new nodes in a breadth first order (always choosing a node with the smallest $h(n)$) with Step 2 as usual.

Theorem 3. *The algorithm as of Def. 5 is complete and sound.*

Proof. It is easy to see that our extension to the pruning rule ensures that Θ and the corresponding node expansion is consistent with what would have been derived when using C_i instead of C_j in the first place. That is, the corresponding missing edge(s) and the potential sub-DAG(s) get constructed accordingly. Considering rule 3(b) we could even remove C_j from *CS*, which in the end would result in a final *CS* that contains only subset-minimal conflicts. Thus, completeness and soundness of HS-DAG-RC follow directly from completeness and soundness of HS-DAG-RC'. □

From an abstract point of view the way our reasoning affects the construction is similar to the reasoning behind Wotawa's idea in HST [Wotawa, 2001]. That is, we aim to avoid the construction of assumption combinations ($h(n)$) in a sub-DAG iff this $h(n)$ would be considered in other reasoning branches anyway. However, our underlying reasoning and the implementation differ significantly from HST.

Besides working on a DAG, our expansion is still based on a pruned version of a node's label (conflict), rather than a range within bounds propagated (and altered) when constructing HST's tree.

In some sense, our reasoning is more similar to a specific scenario in our Boolean algorithm variant that we presented in [Pill and Quaritsch, 2012]. This variant revised Rule 5 (see our brief description in the preliminaries) such that we consecutively choose as e the elements in a single conflict (for specific details please refer to Lemma 2 in [Pill and Quaritsch, 2012]). When iterating over a single $C \in \mathit{CS}$, these consecutive decisions the Boolean algorithm makes regarding the split elements reflect the structure of Θ for the various branches when HS-DAG-RC expands the same C. For the example in Figure 1, that is for $\mathit{CS}' = \{a, b, c, d, e\} \cup \mathit{CS}$, our Boolean algorithm variant would construct the formula $a \wedge H(\mathit{CS}^a) \vee b \wedge H(\mathit{CS}^b_a) \vee c \wedge H(\mathit{CS}^c_{a,b}) \vee d \wedge H(\mathit{CS}^d_{a,b,c}) \vee e \wedge H(\mathit{CS}^e_{a,b,c,d})$, where $\mathit{CS}^e_{a,b,c,d}$ means that we pruned from the conflicts in *CS* elements a, b, c, d and take the subset of pruned conflicts that is not hit by e. This perfectly resembles the effects of Θ_C as illustrated in Figure 1 and the edge label's inclusion in $h(n)$. While this is an interesting analogy specifically regarding the evaluation, please let us remind you at this point that the Boolean algorithm requires *CS* to be computed a priori, while HS-DAG (including our variant) can operate also on-the-fly, besides other important differences (a prominent one is the pruning).

4 Experimental Results

As reference implementation for HS-DAG, we used our Python implementation (CPython 2.7.1) that we used also for the diagnosis algorithm performance comparison in [Nica *et al.*, 2013]. For our HS-DAG-RC variant we adopted the implementation accordingly. A small change we made for both algorithms is the encoding of the work-list, that is the list of nodes to be processed. As evident in its definition, HS-DAG-RC might construct nodes with an $|h(n)|$ smaller than that of the currently expanded node n, due to the updates regarding Θ in the pruning rule. For an easily manageable node-processing list, we thus use a work-list where nodes are grouped by their $|h(n)|$. Hence, a node with the smallest $h(n)$ can be retrieved easily without the need to sort the list when adding a new node (as we would have to do for a monolithic list). As we experienced no penalty for the HS-DAG variant when using the same grouped list (a list of lists) instead of the original single monolithic list, the reported results for HS-DAG also use this type of worklist.

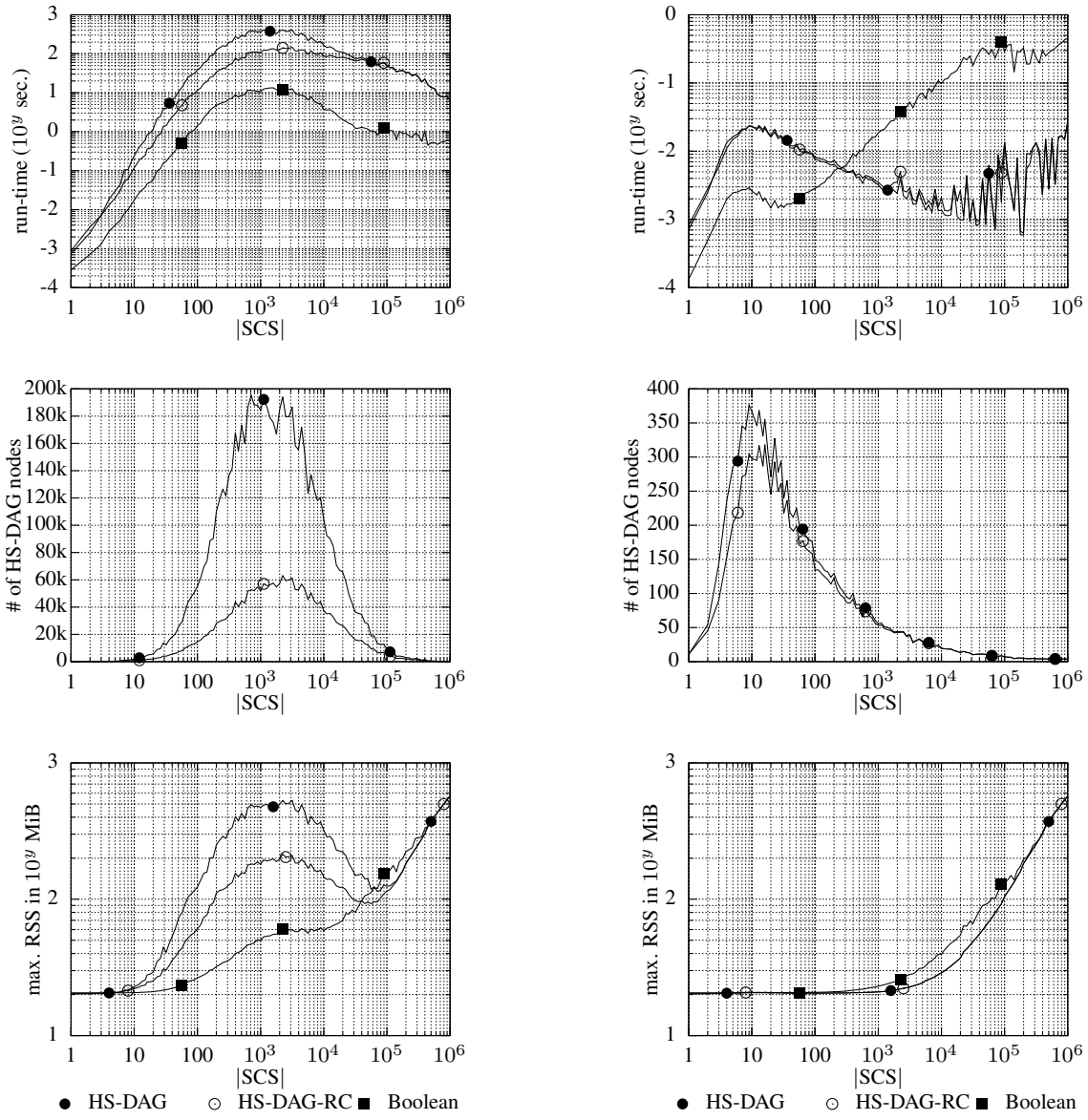
In the following, we report on results for two different scenarios. The first, artificial one we used also for evaluating our optimizations to the Boolean algorithm in [Pill and Quaritsch, 2012] (named TSA1 there). In this scenario, a $C \in \mathit{CS}$ contains elements drawn randomly from *COMP* such that every component $c_i \in \mathit{COMP}$ is included in some $C_i \in \mathit{CS}$ with a probability of 50 percent (no duplicate Cs in *CS* allowed).

Figure 2 reports on the run-time, node-amount and memory consumption (maximum resident size) regarding our experiments with $|\mathit{COMP}| = 20$, and a growing *CS*. We aimed at approximately 120 points equally distributed on the x-axis, which due to rounding resulted in 110 different values for $|\mathit{CS}| \approx 10^{6i/120}$ and $0 \leq i \leq 120$. For each such *CS*, we derived 10 samples and report the average values over the results for the individual samples. Furthermore, we ran both a bounded ($|\mathit{MHS}| \leq 3$) and an unbounded search for each sample. The results of the unbounded search are reported in Figure 2a, those for the bounded one in Figure 2b. Let us consider the unbounded search first. Regarding a comparison between HS-DAG and our variant HS-DAG-RC, we see a run-time advantage for the majority of our $|\mathit{CS}|$ range, with performance on par otherwise. For a $|\mathit{CS}| = 1000$ we experienced a run-time reduction of 70.6 percent, a node reduction of 71.7 percent and a reduction regarding max. RSS of approximately 61.6 percent. The maximum average reductions for any $|\mathit{CS}|$ were 73.5 percent, 75.9 percent and 63.3 percent respectively. For the bounded case we see virtually no difference between HS-DAG and HS-DAG-RC in respect of run-time and memory-consumption. Here the additional computations and variables for Θ seem to counterbalance the slight reduction in the number of nodes (18.9 percent for $|\mathit{CS}| = 10$). As the "pruning"-effect of Θ should increase with the DAG-depth, this is not entirely unexpected for this low cardinality limit of 3, that is, however, often a reasonably low bound in practice. Evidently, in those cases where the Boolean algorithm outperforms HS-DAG, HS-DAG-RC reduces the gap, but is closer to HS-DAG than to the Boolean algorithm.

Our second test scenario is based on conflicts created during specification diagnosis runs as described in [Pill and Quaritsch, 2013] (similar to those for Figure 1(b) in that paper). That is, briefly described, for some specification length in $\{50, 100, ..., 300\}$ we derived 10 random specifications φ in the Linear Temporal Logic (LTL) [Pnueli, 1977] as suggested in [Daniele *et al.*, 1999] with $N = \lfloor |\varphi|/3 \rfloor$ variables and a uniform distribution of LTL operators. We introduced triple faults as described in [Pill and Quaritsch, 2013] in order to derive φ_m from φ. Using the encoding from that paper we retrieved then an assignment for $\tau \wedge \varphi \wedge \neg\varphi_m$ that defines a variable trace τ of length $k = 100$ and loop-back time step $l = 50$. We then solved the diagnosis problem $E(\varphi_m, \tau)$ for a cardinality limit of 3 and recorded the conflicts derived.

Figure 3 reports on the run-time, node amount and memory consumption (maximum resident size) regarding our experiments with the *CS*s recorded for specification diagnosis runs. Again we ran both an unbounded and a bounded ($|\mathit{MHS}| \leq 3$) search for minimal hitting sets. While we observed a run-time reduction of 55.9 percent for $|\varphi| = 300$ in the unbounded case, the computation of Θ seems to entail a slight performance drawback (1.6 milliseconds instead of 1.3 milliseconds) for small samples with $|\varphi| = 50$. Nevertheless, we see also a significant reduction in the number of nodes, that is a reduction of 56.5 percent for $|\varphi| = 300$. The node reduction is however accompanied by an increase in the memory consumption from 40.4 MiB to 54.1 MiB, presumably related to managing Θ. An implementation optimized for low memory consumption could however drop the sets Θ_C and Θ_N after the construction of a subtree and recompute them if needed during a pruning/reconstruction step. The Boolean algorithm outperformed both, HS-DAG and our variant.

Like for the artificial scenario, in the bounded case we see virtually no difference in the run-time, and the reduction in the number of nodes (41.2 percent for $|\varphi| = 300$) is outweighed regarding memory consumption by the needs for Θ



(a) Unbounded

(b) Bounded, $|MHS| \leq 3$

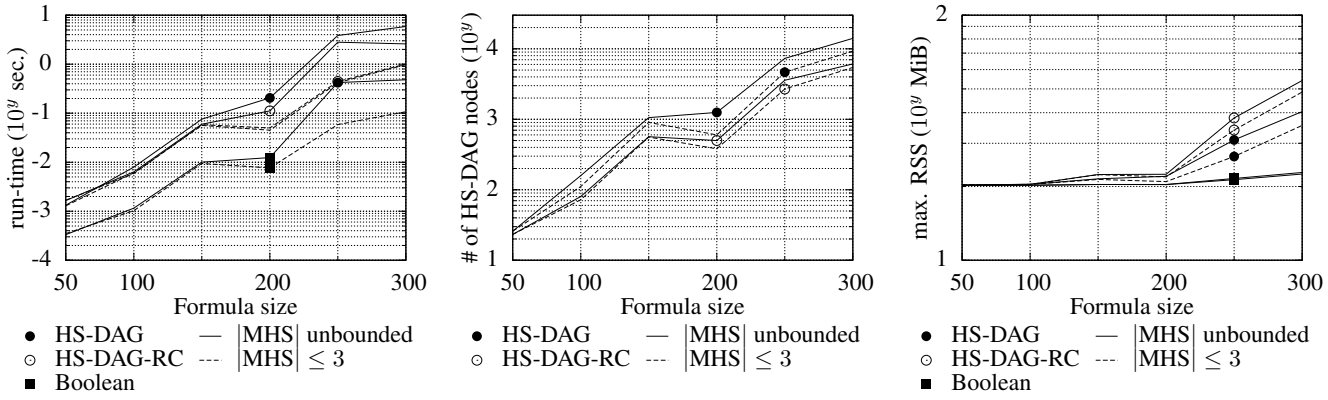
Figure 2: Performance results using random conflicts.

so that with 36.8 percent we have a similar memory penalty for $|\varphi| = 300$ as in the unbounded case (33.9 percent).

Summing up the reported results, we see an attractive performance advantage for our HS-DAG-RC against HS-DAG, specifically for the unbounded *MHS* search. For very small cardinality limits like 3 the still noticeable effects from the node reduction can be diminished by the needs for maintaining Θ, so that we end up with virtually no difference in the run-times but occasionally even experience a penalty in the memory consumption (for the LTL samples we had a penalty, while there was none for the random samples). Thus we see no reason why not to prefer our variant given the reductions in the run-time, node number, and memory consumption (by 73.5 / 75.9 / 63.3 percent, respectively by factors 3.77 / 4.15 / 2.72, for the random samples in the unbounded search) that we could achieve during our tests.

5 Discussion and Conclusions

Previous experiments [Pill *et al.*, 2011; Pill and Quaritsch, 2012] showed that the Boolean algorithm [Lin and Jiang, 2003] is a performant contender when striving for the computation of the complete set of minimal hitting sets. On the other hand it is missing an important feature present in HS-DAG: search space exploration guidance in respect of conflict set computation. As we found it intriguing that this algorithm can iteratively partition and prune the search space, we aimed at a similar concept for HS-DAG. That is retaining HS-DAG's guiding feature that enables an on-the-fly computation of *CS*, our variant HS-DAG-RC mimics the Boolean's divide-and-conquer strategy by removing already considered elements from the (sub-)problem at hand. As we saw, our updated node-expansion routine requires also a more complex pruning-/update function that ensures the tree's compliance with our conflict pruning rules regarding



(a) Run-time (b) Number of HS-DAG nodes (c) Maximum resident set size

Figure 3: Performance results using conflicts from LTL specification diagnosis.

the exclusion-set Θ.

Performance-wise, we could observe in our experiments benefits in the run-time and memory consumptions for our random and LTL specification diagnosis samples. Internally, the nodes constructed for the DAG could be reduced significantly.

While for bounded runs (i.e. limiting the maximum cardinality of solutions to some $|MHS|_{\max}$) our strategy occasionally induced some (minimal) performance and memory overhead compared to the simpler HS-DAG strategy, we expect the experienced benefits to grow with rising maximum cardinality. For unbounded searches computing *all* solutions to a given problem, the experienced savings could be as high as 50–70 percent regarding the run-time, 60–75 percent regarding the number of DAG nodes, and approximately 60 percent in respect of the memory consumption (maximum resident set size).

Future work will investigate the memory penalty occasionally experienced for low bounds in the search, as well as the impact of the more complicated pruning rule 3(a).

Acknowledgements

This work was supported by the Austrian Science Fund (FWF): P22959-N23 ("MoDiaForTeD"). The authors would like to thank Franz Wotawa for fruitful discussions and the anonymous reviewers for their valuable comments.

References

[Daniele *et al.*, 1999] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for Linear Temporal Logic. In *Computer Aided Verification*, pages 249–260, 1999.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[de Kleer, 2011] J. de Kleer. Hitting set algorithms for model-based diagnosis. In *22nd Int. Workshop on the Principles of Diagnosis*, pages 100–105. 2011.

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

[Lin and Jiang, 2003] L. Lin and Y. Jiang. The computation of hitting sets: review and new algorithms. *Information Processing Letters*, 86:177–184, 2003.

[Nica *et al.*, 2013] I. Nica, I. Pill, T. Quaritsch, and F. Wotawa. The route to success - a performance comparison of diagnosis algorithms. In *International Joint Conference on Artificial Intelligence*, pages 1039–1045, 2013.

[Pill and Quaritsch, 2012] I. Pill and T. Quaritsch. Optimizations for the Boolean approach to computing minimal hitting sets. In *20th European Conference on Artificial Intelligence*, pages 648–653, 2012.

[Pill and Quaritsch, 2013] I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *International Conference on Artificial Intelligence*, pages 1053–1059, 2013.

[Pill *et al.*, 2011] I. Pill, T. Quaritsch, and F. Wotawa. From conflicts to diagnoses: An empirical evaluation of minimal hitting set algorithms. In *22nd Int. Workshop on the Principles of Diagnosis*, pages 203–210, 2011.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artif. Intelligence*, 32(1):57–95, 1987.

[Stern *et al.*, 2012] R. Stern, M. Kalech, A. Feldman, and G. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *26th AAAI Conference on Artificial Intelligence*, 2012.

[Wotawa, 2001] F. Wotawa. A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, 79:45–51, 2001.

Finding All Diagnoses is Redundant

Roni Stern[1] and **Meir Kalech**[2] and **Alexander Feldman**[3] and **Shelly Rogov**[2] and **Tom Zamir**[2]
[1]Harvard University, Cambridge, MA USA, e-mail: roni.stern@gmail.com
[2]Ben-Gurion University of the Negev, Beer-Sheva, Israel
e-mail: kalech@bgu.ac.il, shellyr4u@gmail.com, tom.zamir.i@gmail.com
[3] University College Cork, Cork, Ireland, e-mail: alex@llama.gs

Abstract

Consistency-based diagnosis algorithms seek for explanations to unexpected observations in systems. In particular, given a formal model of a system and an observation, a diagnosis is a set of assumables over the health of the components that is consistent with the model and the observation. Consistency-based algorithms are widely researched in the model-based diagnosis literature. A known limitation of consistency-based algorithms is that the number of diagnoses can be very large. Presenting a very large number of diagnoses to a human operator is not likely to be helpful. This raises two questions: (1) how to present the information represented by the consistency-based diagnoses to a human operator? and (2) are all diagnoses required to present this information?. We argue that a human operator requires an aggregated view of the list of partial set of diagnoses, and propose a natural way to do so called the Component Fault Probability (CFP). A CFP shows for every component the probability that it is faulty. Then, we describe a way to evaluate the quality of a CFP, and show that returning all subset minimal diagnoses is seldom required to generate an accurate CFP.

1 Introduction

A diagnosis problem arises when a system does not behave as expected. The goal in diagnosis algorithm is to find the set of system components that caused the unexpected behavior of the system. There are many real-life instances of the diagnosis problem and solving the diagnosis problem has been researched in the Artificial Intelligence community for several decades. One of the fundamental approaches to diagnosis is Model-Based Diagnosis (MBD). In MBD, a formal model of the diagnosed system is assumed to exist that specifies the expected behavior of the system. This model, along with the observed behavior is then used to deduce candidate *diagnoses*. Consistency-based MBD algorithms, a predominant class of MBD algorithms, aim to return all candidate diagnoses that are logically consistent with the system model and the observed system behavior [Reiter, 1987b; de Kleer and Williams, 1987a]. Such candidates are called *diagnoses*.

If there is only a single diagnosis, then this diagnosis is returned by the diagnosis algorithm and the user can then replace or fix the faulty components. Unfortunately, in systems that are not very small it is often the case that the number of diagnoses is very large, potentially exponential in the number of system components. Prior work have proposed algorithms for suggesting additional tests to narrow the list of diagnoses [Shakeri *et al.*, 2000; Feldman *et al.*, 2010b], or for positioning probes to view the output of internal system components [de Kleer and Williams, 1987a]. We consider a different presentation, where the output of the diagnosis algorithm should be displayed to a human operator that will decide on subsequent actions.

Clearly, a human operator cannot intelligently consider a list of hundreds of possible diagnoses. Given the probability of each diagnosis to be correct, one may consider displaying only the most probable diagnoses. In the absence of probabilities, one common approach is to display only diagnoses with the smallest number of components, known as the minimal cardinality diagnoses. Another common approach is to return only a diagnosis that is not a superset of another diagnosis. A set of such diagnoses is known as minimal subset diagnoses. Unfortunately, for systems with hundreds of components or more, the number of even only the minimal cardinality diagnoses becomes so large that even enumerating them is time consuming [Siddiqi, 2011]. Beyond the computational problems, there is an additional user interface challenge: how is a human operator expected to reason about a long list of possible diagnoses?. Furthermore, it is unknown how much information is lost by considering only the highest probable/minimal diagnoses. Figure 2, which is explained in greater detail later in this paper, shows an example where returning the most probable diagnosis is very misleading.

Based on recent research [Feldman *et al.*, 2013; Stern *et al.*, 2012], we consider a natural aggregation of the set of diagnoses by mapping it to a probability of fault for every component. We call the assignment of fault probability to each component the Component Fault Probability (CFP). CFPs can be evaluated by measuring the vector distance between a CFP and the "optimal" CFP produced by the real faults. Among the contributions of this paper is the definition of a CFP, describing how to generate a CFP from a set of diagnoses and how to evaluate a given CFP.

The question we then raise is which and how many diagnoses a diagnosis algorithm needs to find to produce a high quality CFP. Interestingly, we show empirically that finding all minimal subset diagnoses is not needed to produce a good CFP. In fact, improvement caused by adding more

diagnoses to the CFP quality becomes negligible after finding a relatively small number of minimal subset diagnoses, as the CFP quality converges to a specific quality. We also show that the amount of diagnoses required to converge to that CFP quality greatly depends on the order by which the minimal subset diagnoses are found. Convergence of the CFP quality was faster and more stable when the diagnoses were found in order of increasing cardinality than in the reverse order. This result poses a broader question to the diagnosis community: how to search for diagnoses such as to find diverse set of diagnoses that will result in rapid convergence of the CFP quality.

2 Consistency-Based Model Based Diagnosis

Model Based Diagnosis (MBD) problems arise when the normal behavior of a system is violated due to faulty components as indicated by certain observations. We focus on *weak fault models* (WFM), which ignore the mode of abnormal behavior of components [de Kleer *et al.*, 1992].

An MBD problem is specified as a triplet $\langle SD, COMPS, OBS\rangle$ where: SD is a system description, $COMPS$ is a set of components, and OBS is an observation.

The system description takes into account that some components might be abnormal (faulty). This is specified by an unary predicate $h(\cdot)$ on components such that $h(c)$ is true when component c is healthy, while $\neg h(c)$ is true when c is faulty. Denoting the correct behavior of c as a propositional formula, φ_c, SD is given formally as

$$SD = \bigwedge_{c \in COMPS} h(c) \Rightarrow \varphi_c$$

Namely, each component which is healthy follows its correct behavior. A diagnosis problem (DP) arises when, under the assumption that none of the components are faulty, there is an inconsistency between the system description and the observations [de Kleer and Williams, 1987b; Reiter, 1987a].

Definition 1. [Diagnosis Problem]. *Given an MBD problem, $\langle SD, COMPS, OBS\rangle$, a diagnosis problem arises when*

$$SD \wedge \bigwedge_{c \in COMPS} h(c) \ \wedge OBS \vdash \bot$$

For example, a diagnosis problem arises for the MBD of Figure 1 as normal behavior would give output $E = 1$. Once there is an inconsistency, a diagnosis algorithm tries to find a subset $\Delta \subseteq COMPS$ which, if assumed faulty, explains the observation.

Definition 2. [Diagnosis] *Given an MBD problem, $\langle SD, COMPS, OBS\rangle$, the set of components $\Delta \subseteq COMPS$ is a* diagnosis *if*

$$SD \wedge \bigwedge_{c \in \Delta} \neg h(c) \ \wedge \bigwedge_{c \notin \Delta} h(c) \ \wedge \ OBS \nvdash \bot$$

We say that Δ is a minimal diagnosis *if no proper subset $\Delta' \subset \Delta$ is a diagnosis, and that Δ is a* minimal cardinality diagnosis *if no other diagnosis $\Delta' \subseteq COMPS$ exists such that $|\Delta'| < |\Delta|$.*

For the MBD of Figure 1, Δ_1=$\{X_1, X_2\}$, Δ_2=$\{O_1\}$, Δ_3=$\{A_2\}$ are minimal diagnoses, and Δ_2, Δ_3 are minimal cardinality diagnoses, as there is no smaller diagnosis.

Minimal subset diagnoses are especially in the interest of an MBD engine since based on Definition 2 every superset

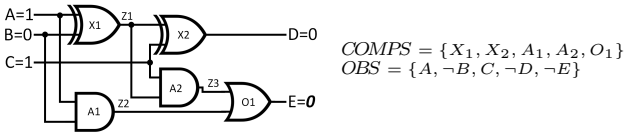


Figure 1: MBD: A full adder.

of a minimal subset diagnosis is a diagnosis too. Therefore, when focusing on the minimal subset diagnoses we actually represent the superset space of those diagnoses. The number of subset minimal diagnoses is typically huge, and therefore a reduced set that contains only the minimal cardinality diagnoses is in the interest of an MBD engine too. Those diagnoses are considered since with no information about the fault probability of the components we can assume identical probability for all the components and thus minimal cardinality diagnoses represent actually those sets with the highest probability.

Beyond the fact that the number of minimal subset and cardinality diagnoses is exponential, MBD is known to be a hard problem where algorithms have exponential runtime (in the size of the system). In terms of complexity, computing the first minimal subset diagnosis is in P, but computing the next one is NP-hard [Bylander *et al.*, 1991]. Computing the minimal cardinality is NP-hard, even for the first diagnosis [Selman and Levesque, 1990]. Both reasons, the exponential number of minimal diagnoses and the exponential computation, motivate an MBD engine to return a reduced set of diagnoses.

An additional challenge is raised as a result of computing a complete set of minimal subset or cardinality diagnoses: how to prioritize the diagnoses? One way to prioritize the diagnoses is by the prior fault probability of the components. With the independent assumption, the probability of a diagnosis can be determined by the product of the components it contains. With no prior probability the diagnoses can be prioritized based on their cardinality.

There are two methods to discriminate the actual diagnosis, the diagnosis that actually contains the faulty components, either by testing or probing [de Kleer and Williams, 1987b]. In the testing method the diagnosis process is run through additional input vectors. Under the assumption that faulty components in the system remain permanently faulty along different input vectors, we can prune diagnoses that are inconsistent with multiple observations. The probing task is similar, but instead of running the diagnosis on a new input vector, the probes are requests on the observation of the output of internal components. Probes can prune diagnoses that are not consistent with the new internal observation. Both methods can be executed iteratively until a single diagnosis is found.

The main challenge in both methods is to reduce the number of probes (tests). A common greedy approach to address this challenge is to choose a probe (test) that maximizes the information gain [Feldman *et al.*, 2010b]. Specifically, given the probability of each diagnosis in the diagnoses set we can measure the entropy of the diagnoses set. The information gain is the difference between the entropy of the diagnoses sets before and after activating a probe (test).

The testing and probing processes may be expensive in the number of probes (tests) required to focus on the actual diagnosis. Furthermore, in some cases the output of a diagnosis algorithm should be displayed to a human operator. A

	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	CFP
C1	1	0	0	0	0	0	**0.2**
C2	1	0	0	0	0	0	**0.2**
C3	0	1	1	1	1	0	**0.8**
C4	0	1	0	0	0	0	**0.16**
C5	0	0	1	0	0	0	**0.16**
C6	0	0	0	1	0	0	**0.16**
C7	0	0	0	0	1	0	**0.16**
C8	0	0	0	0	0	1	**0.16**
p	0.2	0.16	0.16	0.16	0.16	0.16	

Table 1: Diagnoses and CFP for Figure 2

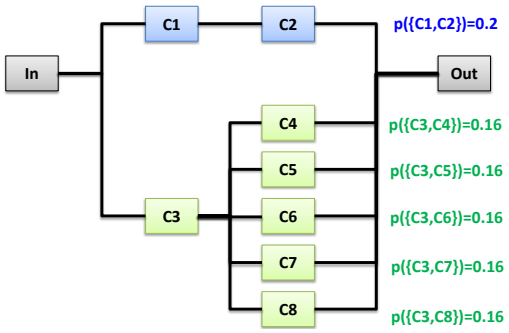


Figure 2: An example where viewing the most probable diagnosis can be more misleading than viewing the CFP. The most probable diagnosis is $\{C1, C2\}$, having a probability of 0.2. The CFP, however, would show that $C3$ has a 0.8 probability of being faulty, and is therefore more likely to be faulty than $C1$ and $C2$.

reasonable question that a user might raise is "what is the probability that a component C is faulty?". This is helpful, for example, to decide which component should be replaced first. More generally, a human diagnosis algorithm operator may wish an estimate of the probability that each component is faulty. Next, we discuss this question and its implications on how diagnosis algorithms should run.

3 Component Fault Probability (CFP)

We use the term Component Fault Probability (CFP) to denote a mapping of components to an estimate of the probability that they are faulty.

Definition 3. [Component Fault Probability (CFP)] *A CFP is a mapping* $COMPS :\rightarrow [0, 1]$ *intended to estimate the probability that a given component is faulty.*

A similar notion to the CFP was introduced in prior work [Stern *et al.*, 2012; Feldman *et al.*, 2013]. Following these prior work, a CFP can be generated from a set of diagnoses that were found by a diagnosis algorithm (DA) as follows.

Let Ω be a set of diagnoses found by a DA, and let $p : \Omega \rightarrow [0, 1]$ be a probability distribution over the diagnoses in Ω, corresponding to the probability that each diagnosis is correct. Many diagnosis algorithms generate this, for example, by considering a prior probability (without considering the observations) on the fault of each component and considering the probability of a diagnosis Δ (denoted $p(\Delta)$) as the product of the prior probability of the components in Δ, i.e., $p(\Delta) = \prod_{C\in\Delta} p(C)$, where $p(C)$ is the prior probability that C is faulty.[1]

A CFP can be generated from Ω and p as follows:

$$CFP(C) = \sum_{\Delta\in\Omega} p(\Delta) \cdot \mathbb{1}_{C\in\Delta} \qquad (1)$$

where $\mathbb{1}_{C\in\Delta}$ is the indicator function defined as:

$$\mathbb{1}_{C\in\Delta} = \begin{cases} 1 & C \in \Delta \\ 0 & \text{otherwise} \end{cases}$$

This way to generate CFP from Ω and p is correct if p assigns correct and independent probabilities to the diagnoses in Ω. Thus, we only consider this way to generate a CFP from Ω and p in this paper, and refer to this process simply as *generating a CFP from* Ω *and* p.

Our main argument in this paper is that presenting a human operator with a CFP is more meaningful and helpful than a long list of diagnoses. As mentioned above, consistency-based DAs may return a very large number of diagnoses. A human operator cannot reason effectively about a long list of diagnoses. One might consider presenting a short list of only the most probable diagnoses to the operator. This approach, however, may be misleading. For example, consider the system depicted in Figure 2, and assume that a DA has returned five diagnoses $\{C3, C4\}, \{C3, C5\}, \{C3, C6\}, \{C3, C7\}, \{C3, C8\}\}$ with a probability of 0.16 each, and another diagnosis $\{C1, C2\}$ with a probability of 0.2[2]. The most probable diagnosis is $\{C1, C2\}$, having a probability of 0.2. The CFP, however, would point at $C3$ as the component that is most likely to be faulty, having $CFP(C3) = 0.8$.

The CFP poses an informative aggregation of a set of diagnoses. However, a set of diagnoses contains more information than the CFP that is generated from it. The information that is lost is the dependency between the different components. For example, consider again the diagnoses in Figure 2. $C1$ only exists in a single diagnosis $\{C1, C2\}$. Thus, $C1$ is only faulty if component $C2$ is faulty as well. This relation is lost in a CFP. Automated algorithms for deciding subsequent tests or probes [Feldman *et al.*, 2010b; de Kleer and Williams, 1987b; Shakeri *et al.*, 2000] might make use of this additional relation information and might thus prefer as input a list of diagnoses over just the CFP. In this paper we focus on diagnosis algorithms whose output is displayed to a human operator. A human operator would find it difficult to reason about a list of diagnoses and would benefit from the aggregated view provided by a CFP.

3.1 Evaluating CFPs

Using Equation 1, one can generate a CFP from a list of diagnoses generated by any MBD algorithm. To evaluate the quality of a generated CFP, we consider the "optimal" CFP, denoted by CFP* and defined as follows:

$$CFP^*(C) = \begin{cases} 1 & C \text{ is faulty} \\ 0 & \text{otherwise} \end{cases}$$

CFP* can be viewed as an *offline* optimal CFP, and a CFP can be evaluated by comparing how "close" it is to CFP*. One can think of many possible distance metrics to measure

[1] To verify that p is a valid probability distribution, one is also required to normalized their sum to one.

[2] The probability of the diagnoses is normalized and therefore their sum is equal to 1.

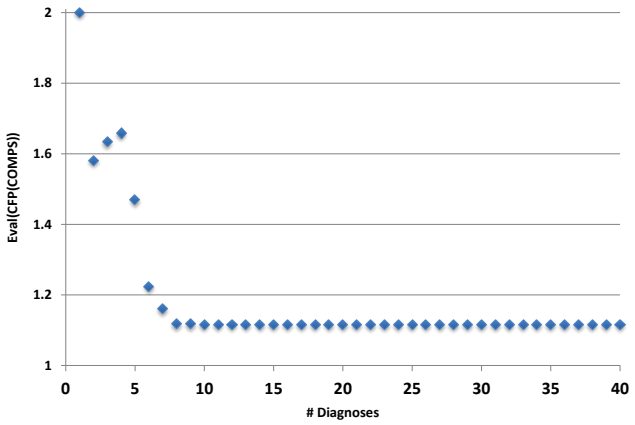


Figure 3: An example where adding more diagnoses improves the CFP quality.

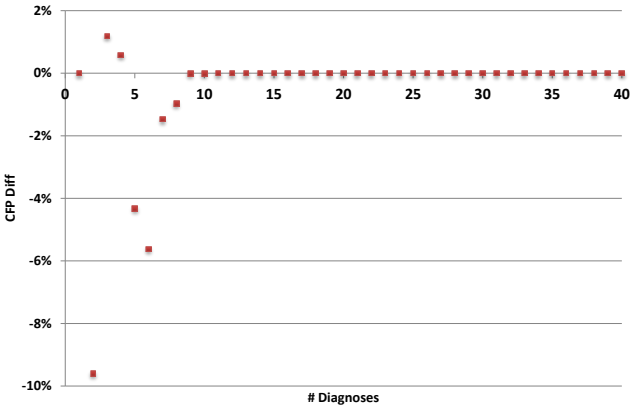


Figure 4: An example showing how the CFP diff converges to zero as more diagnoses are considered.

the "closeness" of a given CFP and CFP*. We chose to use a simple vector distance:

$$Eval(CFP(COMPS)) = \sqrt{\sum_{C \in COMPS}(CFP(C) - CFP^*(C))^2} \quad (2)$$

Lower distance indicates a better CFP. Eval(CFP(COMPS))=0 indicates a diagnosis set that contains only a single diagnosis of faulty components solely.

Introducing CFPs and a way to evaluate them (Equation 2) raises the challenge of constructing diagnosis algorithms that generate quickly high quality CFPs. Next, we evaluate the quality of the CFPs generated from a set of subset minimal diagnoses and the impact of finding more diagnoses to the resulting CFP.

4 Experimental Results

We performed experiments on the 74xxx benchmark Boolean circuits, using observations from the "synthetic track" in the annual DXC diagnosis competition of 2009.[3]

For each observation we run a breadth first search to find all minimal subset diagnoses until either all minimal subset diagnoses were found or until 40 minimal subset diagnoses were found.[4] The resulting set of minimal subset diagnoses were then ordered by cardinality, starting from the diagnoses with the minimal cardinality diagnoses. Let Γ_{inc} be this sequence of diagnoses, where $\Gamma_{inc}[i]$ denotes the i^{th} diagnosis in the sequence.

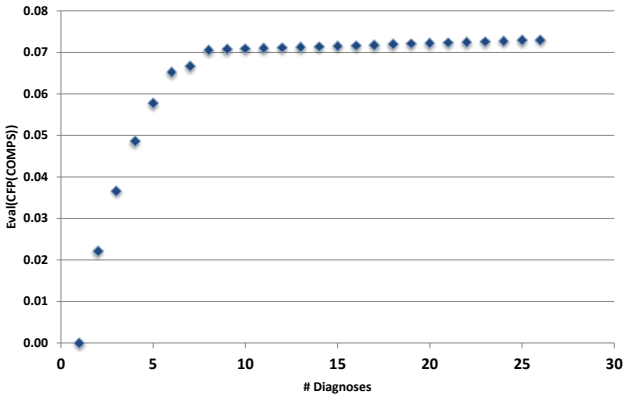


Figure 6: An example where adding more diagnoses only degrades the CFP quality.

For every diagnosis in Γ_{inc} we computed the CFP based on the union of that diagnosis and all the diagnoses that preceded it in Γ_{inc}. Thus, the first CFP was computed based on the diagnosis set that contains only a single diagnosis $\Gamma_{inc}[1]$, and then the second was computed based on a diagnosis set that contains $\{\Gamma_{inc}[1], \Gamma_{inc}[2]\}$, etc. Let $CFP_{inc}[i]$ denote the CFP generated from $\{\Gamma_{inc}[1], \Gamma_{inc}[2], ..., \Gamma_{inc}[i]\}$ and let $Eval(CFP_{inc}[i])$ denote its quality (measured using Equation 2). For every $CFP_{inc}[i]$ we measured the difference in CFP quality between two subsequent CFPs (using Equation 2) and normalize the result by dividing it with the square root of the number of components in the diagnosed system. We call this measure "CFP diff", computed for the i^{th} diagnosis in Γ_{inc} as follows:

$$\frac{Eval(CFP_{inc}[i]) - Eval(CFP_{inc}[i-1])}{\sqrt{|COMPS|}} \quad (3)$$

CFP diff is intended to measure the improvement of CFP by searching for additional diagnoses. Figures 3 and 4 show the CFP quality and the CFP diff, respectively, as a function of the number of diagnoses found, for a single observation of the 74182 system.

Figure 5 shows the CFP diff as a function of the number of diagnoses used to generate that CFP for all the 74xxx systems and all the observation set in the DXC benchmark set. Each of the red x mark data points correspond to a CFP diff of a specific observation as a function of the number diagnoses seen so far.

As can be seen, the value of adding diagnoses converges very close to zero. This suggests that finding more diagnoses is not always helpful, and that finding all minimal diagnoses is not needed. Thus, a smart diagnosis algorithm might exploit this by finding only some of the minimal diagnoses instead of all them. This is expected to result in a better runtime and practically the same quality of CFP.

[3] See details in the DXC 09 website: `http://sites.google.com/site/dxcompetition2009/`.

[4] 40 was chosen to avoid exhausting memory or time. Other bounds are also possible.

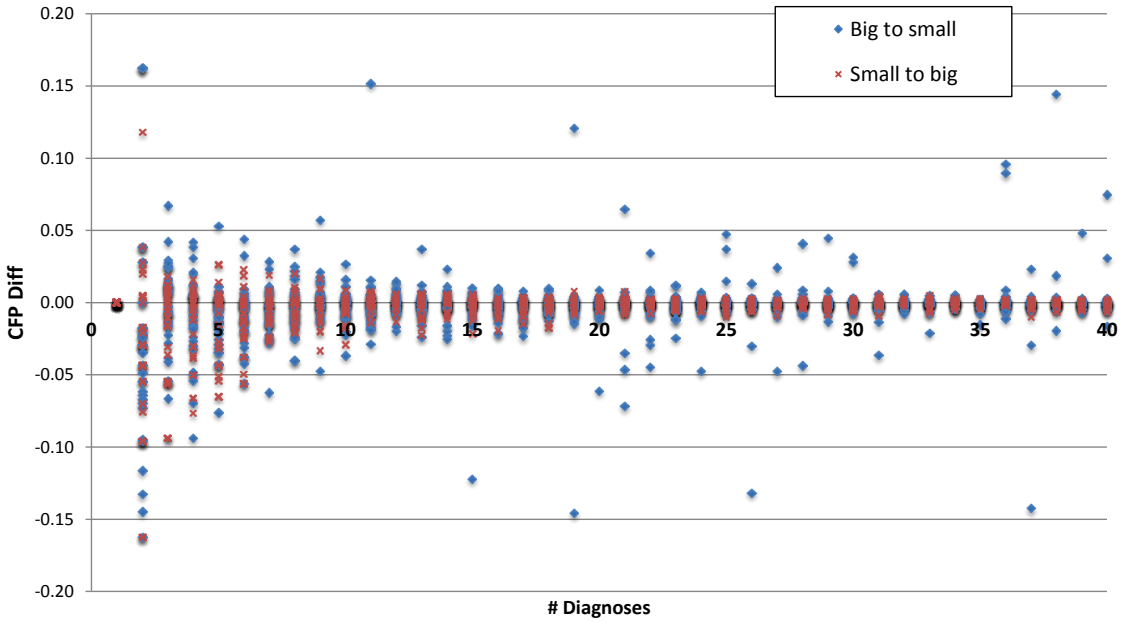


Figure 5: CFP diff as a function of the number of diagnoses found, when searching for subset minimal diagnoses. The red x marks are for experiments where the diagnoses were found in order of increasing cardinality, while the blue diamonds are for when diagnoses where found in the reverse order.

The convergence of the CFP diff to zero, observed in Figure 5, suggests that in every observation the CFP quality converges to a specific value as more consistent diagnoses are found. This can be seen clearly also in Figure 3, where the CFP quality converges to approximately 1.16 as more diagnoses are found. The converged CFP quality is not necessarily better than the CFP quality obtained by considering only a subset of the consistent diagnoses. For example, Figure 6 shows the CFP quality as a function of the number of diagnoses found for a different observation. In this observation, the CFP quality starts at the optimal value (zero distance from CFP^*). This is since the correct diagnosis happened to be the first diagnosis that was found. As more diagnoses are found, the CFP quality degrades. Thus, considering more diagnoses is not always helpful. Since when running a diagnosis algorithm one does not know CFP^*, it is a challenge to identify when adding more diagnoses does not help.

Next, consider the blue diamonds in Figure 5. These diamonds show the CFP diff for the same set of experiments, where the order of the sequence of diagnoses is reversed. This means that the first diagnoses considered is not the minimal cardinality diagnoses, but rather the maximum cardinality diagnoses. As can be seen, while a general trend of convergence to zero exists to some extent, it is much more noisy then when diagnoses were considered in order of increasing cardinality. This demonstrates the potential importance of the order by which diagnoses are found by the diagnosis algorithm.

5 Related Work

Many of the existing diagnosis techniques propose to apply a combination of deterministic reasoning and search algorithms. One classic approach involves a two stage process. First, it identifies conflict sets, each of which includes at least one fault. Then, it applies a hitting set algorithm to compute sets of multiple faults that explain the observation [de Kleer and Williams, 1987a; Williams and Ragno, 2007]. These methods guarantee sound diagnoses, and some of them are even complete. However, they tend to fail for large systems due to infeasible runtime or space requirements.

Compilation-based methods have also been proposed in the MBD context. Torasso and Torta apply BDDs to compile the model [Torasso and Torta, 2006]. Darwiche [Darwiche, 2001] compiles a system description into Decomposable Negation Normal Form (DNNF) where a minimal cardinality diagnosis can be found in time that is polynomial in the size of the DNNF. However, the size of the DNNF may grow exponentially and is shown to become a bottleneck [Siddiqi and Huang, 2007].

Feldman *et al.* [2010a] propose a stochastic diagnosis algorithm called SAFARI. In contrast to the above, SAFARI does not try to compute the set of all diagnoses and it does not guarantee to find minimal cardinality diagnoses. Its advantage is that it is very fast. Like other MBD algorithms, it tries to generate as many small diagnoses as possible and does not consider aggregating the resulting set of diagnoses. Thus, the work in our paper is orthogonal to SAFARI, as a CFP can be generated from the diagnoses found by SAFARI using Equation 1.

Keren *et al.* [2011] present an alternative approach to diagnosis that combines MBD with multi-label classification. They propose to build a classifier that maps symptoms (observations) of the system to possible faults. The major advantage of this approach is in reducing significantly the online computational complexity; The learning process of the relations between observations and the diagnoses is performed in advance offline. Afterwards (online), a diagnosis can be computed immediately by using the classifier that was learned offline. Unlike other diagnosis algorithms mentioned above, this machine learning approach to diagnosis returns a single diagnosis and not a set of diagnosis. Similar

to our approach, the output of this machine learning based diagnosis algorithm is not measured by its consistency but by its distance to the real diagnosis (the faults), using standard classification metrics such as false positives and false negatives. In our work we evaluated the CFP with a simple vector distance metric instead. In future work, however, we plan also to evaluate our results using machine learning metrics such as false positives and negatives, precision and recall.

Maier et al. [2011] pointed out that often AI problems lie at the intersection of the fields of model-based diagnosis and probabilistic reasoning and that probabilistic reasoning can be a promising alternative to the model-based diagnosis approaches. They use a Bayesian networks (BNs) results from first-order model-based diagnosis formalism for this translation to first-order probabilistic reasoning framework. Similar to our approach, they also remark that solutions to AI problems in engineering domains need to be compactly represented for the needs of engineers. However, by compactly represented they mean auto-generating low-level representation such as BNs that can be used as an input to off-the-shelf tools while we aim to find a more general and simple approach for compact representation not by finding the connections and dependencies between components but rather by addressing the logic model of the system done by aggregating the diagnoses to CFP.

6 Conclusion and Future Work

In this work we proposed an alternative form of output to diagnosis algorithm called the *component fault distribution*, or CFP in short. CFP maps every component in the diagnosed system to a probability that this component is faulty. We argue that a CFP is a more reasonable output than a single or k most probable diagnoses since it contains aggregated information over all the found diagnoses. We also argue that a CFP is more reasonable to a human operator than a long list of diagnoses. CFPs can be generated from a list of diagnoses that are returned by a diagnosis algorithm, and a way to evaluate CFPs is proposed. Empirical evaluation on the 74xxx system suggests that the quality of a CFP converges quickly as more diagnoses are found.

This observation leads to considering, in future work, how to develop a diagnosis algorithm that finds a small but representative set of diagnoses that will generate a high quality CFP. One might even consider generating a CFP from "almost" consistent diagnoses - set of components that if assumed to be faulty explains almost all the observed system outputs.

References

[Bylander *et al.*, 1991] Tom Bylander, Dean Allemang, Michael C. Tanner, and John R. Josephson. The computational complexity of abduction. *Artif. Intell.*, 49(1-3):25–60, 1991.

[Darwiche, 2001] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

[de Kleer and Williams, 1987a] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[de Kleer and Williams, 1987b] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

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

[Feldman *et al.*, 2010a] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. *J. Artif. Int. Res.*, 38:371–413, May 2010.

[Feldman *et al.*, 2010b] Alexander Feldman, Gregory Provan, and Arjan van Gemund. A model-based active testing approach to sequential diagnosis. *J. Artif. Int. Res.*, 39:301–334, September 2010.

[Feldman *et al.*, 2013] Alexander Feldman, Helena Vicente de Castro, Arjan van Gemund, and Gregory Provan. Model-based diagnostic decision-support system for satellites. In *Aerospace Conference, 2013 IEEE*, pages 1–14. IEEE, 2013.

[Keren *et al.*, 2011] Betty Keren, Meir Kalech, and Lior Rokach. Model-based diagnosis with multi-label classification. In *The 22nd International Workshop Principles of Diagnosis (DX-11)*, 2011.

[Maier *et al.*, 2011] Paul Maier, Dominik Jain, and Martin Sachenbacher. Diagnostic hypothesis enumeration vs. probabilistic inference for hierarchical automata models. In *the International Workshop on Principles of Diagnosis (DX)*, Murnau, Germany, 2011.

[Reiter, 1987a] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96, 1987.

[Reiter, 1987b] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[Selman and Levesque, 1990] Bart Selman and Hector J. Levesque. Abductive and default reasoning: A computational core. In *AAAI*, pages 343–348, 1990.

[Shakeri *et al.*, 2000] Mojdeh Shakeri, Vijaya Raghavan, Krishna R Pattipati, and Ann Patterson-Hine. Sequential testing algorithms for multiple fault diagnosis. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(1):1–14, 2000.

[Siddiqi and Huang, 2007] Sajjad Ahmed Siddiqi and Jinbo Huang. Hierarchical diagnosis of multiple faults. In *IJCAI*, pages 581–586, 2007.

[Siddiqi, 2011] Sajjad Ahmed Siddiqi. Computing minimum-cardinality diagnoses by model relaxation. In *IJCAI*, pages 1087–1092, 2011.

[Stern *et al.*, 2012] Roni Stern, Meir Kalech, Niv Gafni, Yair Ofir, and Eliav Ben-Zaken. Using model-based diagnosis to improve software testing. In *the International Workshop on the Principles of Diagnosis (DX)*, 2012.

[Torasso and Torta, 2006] Pietro Torasso and Gianluca Torta. Model-based diagnosis through obdd compilation: A complexity analysis. In *Reasoning, Action and Interaction in AI Theories and Systems*, pages 287–305, 2006.

[Williams and Ragno, 2007] Brian C. Williams and Robert J. Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Appl. Math.*, 155(12):1562–1595, 2007.

International Diagnostic Competition

Fourth International Diagnostic Competition – DXC'13

Adam Sweet[1], **Alexander Feldman**[2], **Sriram Narasimhan**[3], **Matthew Daigle**[1], and **Scott Poll**[1]
[1]NASA Ames Research Center, Moffett Field, CA, U.S.A.
email: {adam.sweet, matthew.j.daigle, scott.poll}@nasa.gov
[2]University College Cork, Cork, Ireland
email: alex@general-diagnostics.com
[3]University of California, Santa Cruz @ NASA Ames Research Center, Moffett Field, CA, U.S.A.
email: sriram.narasimhan-1@nasa.gov

Abstract

We present the description and results of the Fourth International Diagnostic Competition, which tests and evaluates diagnostic algorithms (DAs). This year's competition offered the industrial, synthetic and software tracks from previous competitions, and a new thermal-fluid track. Only the industrial track competition was held, with a total of 5 DAs participating. The paper briefly reviews the industrial track used in this and previous competitions. The participating DAs are described, and the scoring metrics and competition results are presented.

1 Introduction

Much research has been done in the field of diagnosis, resulting in many types of algorithms capable of detecting and isolating faults in many types of systems. However, until recently there have been few efforts to evaluate and compare these algorithms in a standard way. NASA Ames Research Center (ARC), Palo Alto Research Center (PARC), and Delft University of Technology decided to combine efforts to create a generalized framework that would establish a common platform to evaluate and compare diagnosis algorithms (Kurtoglu et al., 2009a). The objectives for developing this framework were to accelerate research in theories, principles, and computational techniques for monitoring and diagnosis of complex systems; to encourage the development of software platforms that promise more rapid, accessible, and effective maturation of diagnostic technologies; and to provide a forum that can be utilized by algorithm developers to test and validate their technologies on real-world physical systems.

A series of competitions has been held using the framework to test and evaluate diagnostic algorithms (DAs) in a variety of diagnostic problem domains. The First International Diagnostic Competition (DXC'09) was held under the auspices of the DX conference, and the methods and results of the competition are presented in (Kurtoglu et al., 2009b). In that competition, two tracks were defined to present diagnostic problems in different domains: an industrial track focusing on an electrical power system and a synthetic track focusing on logic circuits. A set of metrics was created to quantitatively evaluate the diagnostic algorithms, and the metrics were weighted to determine the overall winner. However, this approach has the weakness that a DA's score in the competition is heavily dependent on the weights assigned to each metric. In practice, the importance of different diagnostic metrics depends on the requirements of the application.

The scoring method was changed for the Second International Diagnostic Competition (DXC'10), (Poll *et al.*, 2010). Instead of the DXC'09 scoring with the same weights on the metrics for all tracks, the scores for each track were determined according to a use case defined for that track. The industrial track used a decision support use case, where the diagnosis would be used to determine a recovery action. The score for a DA depends on the correctness of the recommended recovery action. The synthetic track used a troubleshooting use case, where many internal variables were not observable and probes must be used to determine an unambiguous diagnosis. The goal was to correctly identify the fault with the fewest probes. The metrics were still calculated and used as tiebreakers.

A Third International Diagnostic Competition (DXC'11) was held, and introduced a new track on software diagnosis (Poll *et al.*, 2011). The goal of this track is to provide common ground to evaluate techniques that diagnose failures in software systems. For DXC'11, the focus was on techniques that use coverage data; the algorithms were evaluated on their performance in finding software faults based only on that coverage data.

Finally, the present Fourth International Diagnostic Competition (DXC'13) was held. The most significant addition to this competition was the new thermal fluid track, which presented problems in a building's heating, ventilation, and air conditioning (HVAC) domain. The tracks from previous years were also available. The industrial track's electrical power system competition was performed, with the same format as in previous competitions and with newly acquired competition data.

The paper is organized as follows. Section 2 gives a quick review of the DXC framework. Section 3 describes the diagnostic problems that were presented to the competitors. Section 4 lists the kinds of faults that were injected. Section 5 explains how the evaluation was performed. Section 6 presents the results. Section 7 concludes the paper.

2 DXC Framework

The DXC framework was developed for DXC'09 and modified in subsequent years. It allows DAs to be tested under identical experimental conditions and saves and evaluates the result of the tests. The key components of this framework include representation languages for the physical system description, sensor data and diagnosis results, a run-time architecture for executing DAs and diagnostic scenarios, and an evaluation component that computes performance metrics based on the results from diagnosis algorithm execution.

The DXC framework has been extensively described in past publications; the reader may refer to (Kurtoglu *et al.*, 2009a; Feldman *et al.*, 2010; and Poll *et al.*, 2011) for those descriptions and architecture diagrams. A textual description of the main run-time components of the framework is repeated here:

Scenario Loader (SL): Executes the Scenario Data Source, Recorder, and Diagnosis Algorithm. SL ensures system stability and clean-up upon scenario completion. This is the main entry point for performing a diagnostic experiment.

Scenario Data Source (SDS): Provides scenario data from previously recorded datasets. The provenance of the data (whether hardware or simulation) depends on the system in question. A scenario dataset contains sensor readings, commands (note that the majority of classical model-based diagnosis literature does not distinguish commands from observations), and fault injection information (to be sent exclusively to the Scenario Recorder). SDS publishes data following a wall-clock schedule specified by timestamps in the scenario files.

Scenario Recorder (SR): Receives fault injection data and diagnosis data into a scenario results file. The results file contains a number of time-series which are used by the evaluation module for the computation of metrics. SR is the main timing authority, i.e., it timestamps each message upon arrival before recording it to the results file.

Diagnosis Algorithm (DA): A DA receives sensor and command data, performs diagnosis, and sends the diagnosis results back. As long as the DAs comply with the provided API, there are no restrictions on a DA; for example, a DA may read precompiled data, or use external (user supplied) libraries, etc.

Diagnostic Oracle: The Diagnostic Oracle is only relevant to the Industrial Track. It provides a querying capability to the DAs in one of two ways: 1) it takes a diagnostic output produced by a DA and returns the lowest cost action(s) associated with the provided diagnosis, or 2) it takes a diagnostic output and specific actions produced by a DA and returns the corresponding cost.

Evaluator: Takes scenario result file and applies metrics to evaluate DA performance. The metrics and evaluation procedures are detailed in Section 5.

3 Diagnostic Problems

Five diagnostic problems were announced for DXC'13: two industrial track problems (DP-I, DP-II), one synthetic (DP-III), one software (DP-IV), and the new thermal-fluid track problem (DP-V). Unfortunately, the only entries received were in DP-I. The other competition tracks will be maintained and hopefully used in future competitions.

The system used for DP-I, ADAPT-Lite, is based on the Electrical Power System (EPS) testbed in the ADAPT lab located at NASA Ames Research Center (Poll *et al.*, 2007). This system and the DP-I problem are described in detail in previous publications (Kurtoglu *et al.*, 2009a; Feldman *et al.*, 2010; and Poll *et al.*, 2011). It has not been changed for DXC'13. A brief summary is presented in this section.

A subset of the components and sensors on the ADAPT EPS is used to mimic the operation of an electrical system aboard a single-string Unmanned Aircraft System (UAS). DP-I does not represent any particular UAS, but may be thought of as a generic UAS carrying instruments that acquire scientific data. A system schematic for DP-I is given in Figure 1. BAT2 is a battery supplying electrical power to several loads in the UAS system. The power is transmitted through several circuit breakers (with component names beginning in "CB") and relays ("EY"), and an inverter INV2 to supply AC power. The loads are named AC483, FAN416, and DC485. There are also sensors throughout the system to report electrical voltage (names beginning with "E"), electrical current ("IT"), and the positions of relays and circuit breakers ("ESH", "ISH"). Finally there is one sensor to report the operating state of a load (fan speed, "ST").

The DA is used for decision-support during the UAS mission to inform the pilot if faults have occurred, and if so, whether the fault requires aborting the mission and landing the UAS. The necessity of an abort depends on the fault present, and in some cases, on the values of the fault parameters. Any failure which cuts off power to one of the three loads requires an abort. The other failures result in a degraded operation, and while some do not require an abort, others may, depending on the fault parameters. Thus, this diagnostic problem requires the DAs to perform fault detection, isolation, and parameter estimation. In the context of the DXC, the DA will be scored according to the correctness of its abort (or no-

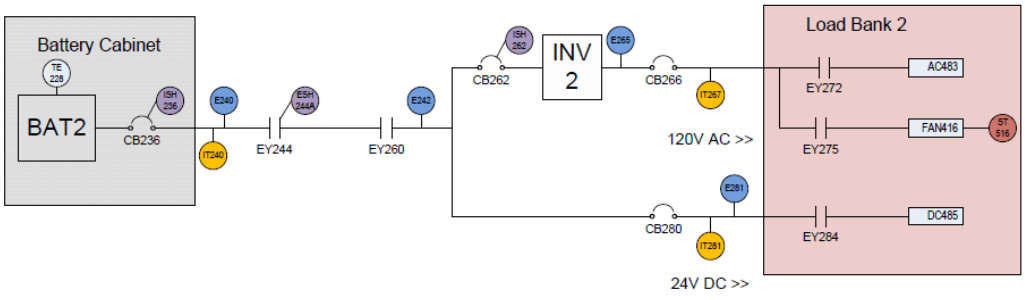


Figure 1: ADAPT-Lite system for DP-I

abort) recommendation as described in later sections.

Finally, with the given sensors, DP-I contains four diagnostic ambiguity groups: (i) AC483 failed off and EY272 stuck open, (ii) FAN416 failed off and EY275 stuck open, (iii) DC485 failed off and EY284 stuck open, and (iv) INV2 failed off and CB262 failed open. In each case however, the recovery action is the same for both faults in the ambiguity group.

4 Fault Injection and Scenarios

The DP-I scenarios are a series of approximately four-minute scenarios gathered from the ADAPT EPS. DP-I scenarios only contain a single injected fault, but DAs may report multiple faults as part of an ambiguity group (especially for the ambiguities listed in section 3.1).

The ADAPT EPS is designed to allow repeatable fault injection, in one of several ways. The first method is directly in the hardware, by manually switching components off or on or manipulating the load resistances. The second method is in software, by intercepting user commands or sending extraneous commands to the EPS, both unbeknownst to the DA. The third method is done with postprocessing of a nominal data run; this method is useful for injecting sensor faults, as they don't affect other components of the system.

For DXC'13, 114 new scenarios were gathered from the ADAPT EPS and used for the competition scenarios. All previous training and competition scenarios were available to entrants to use as training scenarios.

5 Evaluation

This section describes the scoring and the computation resources used in this year's competition.

5.1 Scoring

As described in Section 1, the entries in the DP-I diagnostic problem will be evaluated on the basis of the correctness of their abort recommendations. A cost is assigned to each DA's abort or no-abort recommendation for each scenario, and the total cost for all scenarios is summed to determine the DA's final score. The DA with the lowest cost is the competition winner. The costs are summarized in Table 1. As shown in the table, a correct abort recommendation is given a cost of 0, and an incorrect abort recommendation's cost depends on what should have been recommended for that scenario. If a scenario contains a fault which does not require an abort, but the DA recommends an abort, it is regarded as a loss of mission and given the cost $C_{mission}$. If a scenario contains a fault requiring abort but the DA did not recommend abort, it is regarded as a loss of vehicle, $C_{vehicle,}$ and the mission $C_{mission}$, which is a much higher cost. For DP-I, $C_{mission} = 25$, and $C_{vehicle} = 100$. A perfect-scoring DA will thus have an overall competition cost of 0.

Table 1: DP-I Decision Costs ($\mathbf{M}_{dc}$)

Actual Case / DA Rec.	Abort	Non-abort
Abort	0	$C_{mission}$
Non-abort	$C_{mission}$ + $C_{vehicle}$	0

The metrics used in DXC'09 will also be gathered and calculated for purposes of tie-breaking and comparison. Please see (Kurtoglu et al., 2009; Feldman et al., 2010) for detailed definitions and related discussion. These metrics are summarized in Table 2. Note that DXC'09 metric $\mathbf{M}_{ia}$ has been renamed $\mathbf{M}_{err}$ in the table. The metrics in the table are per scenario metrics. To calculate "per system" metrics an unweighted average is taken over all scenarios and is indicated with an overbar.

5.2 Computing platform

DP-I diagnosis algorithms were evaluated using the DXC framework on a Windows 7 computer with an Intel i7-3770S CPU running at 3.10 GHz.

6 Results

This section describes the entrants into the competition and presents the competition results.

6.1 Diagnosis Algorithms

A description of each DA entered in DXC'13 is given below:

1. QED: A model-based diagnosis system based on qualitative event-based fault isolation. Statistically significant deviations of measured from model-predicted values imply the presence of faults. These deviations are abstracted into symbolic event-based descriptions of fault-induced behavior, which are compared to predicted event sequences to isolate faults. Fault identification uses quantitative methods to compute fault parameters and further refine fault hypotheses (Daigle and Roychoudhury, 2010).

Table 2: DXC'09 Metrics

Metric	Name / description
$\bar{\mathbf{M}}_{fd}$	Fault detection time, average
$\bar{\mathbf{M}}_{fn}$	False negative rate, percentage of total scenarios
$\bar{\mathbf{M}}_{fp}$	False positive rate, percentage of total scenarios
$\bar{\mathbf{M}}_{da}$	Detection accuracy, percentage of total scenarios
$\bar{\mathbf{M}}_{fi}$	Fault isolation time, average
$\mathbf{M}_{err}$	Classification errors, sum of all scenarios
$\bar{\mathbf{M}}_{cpu}$	Computer processing time used, average
$\bar{\mathbf{M}}_{mem}$	Computer memory used, average

2. QED-PC: Similar to QED, but uses the Possible Conflicts diagnosis approach (Pulido and Alonso-Gonzalez, 2004). The global system model is de-composed into minimal submodels containing a sufficient analytical redundancy to generate fault hypothesis from observed measurement deviations. (Daigle *et al.*, 2011).
3. QED-PC++: Combines the residual sets of QED and QED-PC to improve fault isolation over each algorithm individually. Residuals from both the global model and PCs are used for fault isolation within the qualitative fault isolation framework. This improves diagnosability and fault isolation times.
4. HyDE: Hybrid Diagnosis Engine (HyDE) is a model based diagnosis engine that uses consistency between model predictions and observations to generate conflicts which in turn drive the search for new fault candidates. The model is used to simulate system behavior which is compared actual system behavior to identify any discrepancies. The discrepancies are used to guide the isolation of possible fault faults that would make simulated and actual system behavior consistent.
5. HyDE-PC: Similar to HyDE, but also uses the Possible Conflicts approach (Pulido and Alonso-Gonzalez, 2004).

6.2 Results

The metrics for all 5 DAs in this year's competition are summarized in Table 3. The DAs are listed in the table in order of their ranking for the competition, based on the decision cost ($\mathbf{M}_{dc}$) metric.

QED was the winner of the DP-1 competition, with a total cost of 250. This is due to missing 2 aborts. The first missed abort seems to be due to a software glitch. QED produced a correct diagnosis but for some reason did not query the oracle or send a recovery action. The same fault is repeated at a different time in another scenario, and in that other scenario QED does query the oracle and issue the correct recovery action to abort. For the second, the injected fault is quite close to the abort vs. no-abort threshold. While QED's determination of the fault parameters was quite close to the exact values used in the fault injection, those values indicate to not abort. The exact values of the injected fault parameters do count as an abort being the correct action for that scenario.

Some other interesting aspects of the results are seen in other metrics. QED had the best overall diagnosis results, with lower false positive ($\bar{\mathbf{M}}_{fp}$) and higher diagnostic accuracy ($\bar{\mathbf{M}}_{da}$) rates than QED-PC and QED-PC++. This is most likely a result of the maturity of the algorithm: QED has been entered in all of the past competitions, while QED-PC and QED-PC++ are similar but newer. Also, QED-PC++ has much lower fault detection times and fault isolation times than QED or QED-PC. This was the expected behavior, borne out by experimental verification in DXC'13.

The HyDE and HyDE-PC DAs were also not as mature as QED, although HyDE was a previous entrant it has not been entered in every competition. Both variants of HyDE incurred higher decision costs than the variants of QED. Looking at Table 3, the HyDE variants had a bias toward false negatives ($\bar{\mathbf{M}}_{fn}$) rather than false positives ($\bar{\mathbf{M}}_{fp}$). Hence, HyDE was biased toward not commanding an abort. This unfortunately is not a good strategy given the scoring for DP-I; far more cost is assigned to incorrectly losing a vehicle (by failing to abort) than

Table 3: DP-I Competition Results

DA	$\mathbf{M}_{dc}$	$\bar{\mathbf{M}}_{fd}$ (s)	$\bar{\mathbf{M}}_{fn}$	$\bar{\mathbf{M}}_{fp}$	$\bar{\mathbf{M}}_{da}$	$\bar{\mathbf{M}}_{fi}$ (s)	$\mathbf{M}_{err}$	$\bar{\mathbf{M}}_{cpu}$ (ms)	$\bar{\mathbf{M}}_{mem}$ (kb)
QED	250	3.255	0.0	0.035	0.9649	71.861	25	10.8	7504
QED-PC++	1350	1.657	0.0	0.439	0.5614	43.837	66	13.5	7847
QED-PC	1500	3.133	0.0	0.406	0.5965	67.146	85	9.9	7687
HyDE-PC	2550	8.157	0.20	0.018	0.807	8.316	112	430	59279
HyDE	2650	8.769	0.19	0.018	0.8158	8.857	121	1311	83189

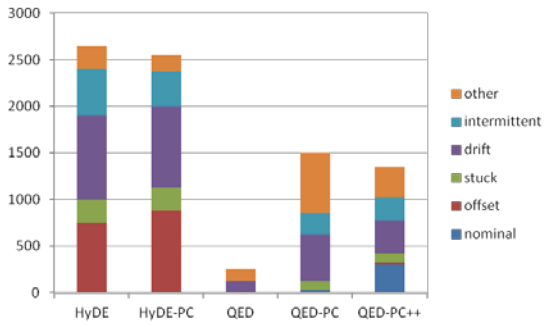


Figure 2: DP-I Cost by scenario type

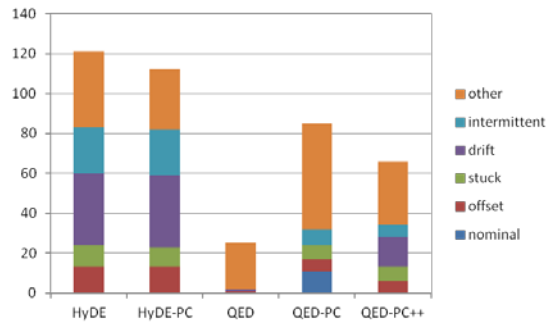


Figure 3: DP-I classification errors by scenario type

to incorrectly losing a mission (by aborting when unnecessary). The HyDE variants had the lowest false positive rates of all the entrants; if the HyDE variants were tuned to report fewer false negatives (even at a cost of increased false positives) it is likely their scores would have been similar to QED-PC and QED-PC++.

We show the breakdown of decision cost ($\mathbf{M}_{dc}$) by fault type for each DA in Figure 2. Offset, drift, and intermittent faults include hardware (AC483, DC485) and sensor (e.g., IT267, IT281, etc.) fault injection scenarios. Category "other" includes circuit breaker, inverter, fan, and AC and DC load failed-off fault scenarios. The QED variants are all good at detecting the "offset" type of fault, as Figure 2 shows: a very small slice (or no slice) is red indicating the "offset" type. The HyDE variants are fairly uniform in their ability to detect different types of faults, and had a considerably higher overall cost due to the immaturity of the diagnoser.

We also show the breakdown of classification errors ($\mathbf{M}_{err}$) by fault type in Figure 3. In a scenario, the number of classification errors is the number of misclassified components. Ruling out guessing, a perfect DA would have 23 classification errors, all in the category "other", because of the ambiguity groups.

7 Conclusion

We presented the implementation of the Fourth International Diagnostic Competition, DXC'13. This year's competition featured 5 DAs each competing in the industrial track problem DP-1. The winner had an excellent score, showing the maturity of the algorithm having been developed and entered in several competitions.

The authors hope that future competitions will garner more interest than this year's competition. The authors also hope that the DXC framework and data are useful to the research community outside of the competition, as a standard means to compare and benchmark diagnosis algorithms. Finally, we continue to hope that the framework is applied to more physical systems and diagnostic algorithms in the future.

Acknowledgments

As the competition and framework have been developed over many years, the authors would like to acknowledge the contributions of all previous DXC organizers and developers of the DXC framework, in particular Tolga Kurtoglu (PARC), David Garcia (PARC), and Johan De Kleer (PARC). We would also like to thank Amarnath Raveendran (Georgia Tech) and David Nishikawa (NASA) for their efforts in collecting new competition data for DP-I.

References

[Abreu et al. 2009] R. Abreu, P. Zoeteweij, A.J.C. van Gemund. A New Bayesian Approach to Multiple Intermittent Fault Diagnosis. In Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI'09), pp. 653-658, Pasadena, CA, USA, July 2009.

[Abreu et al., 2011] R. Abreu, P. Zoeteweij, A.J.C. van Gemund. Simultaneous Debugging of Software Faults. In Journal of Systems and Software (JSS), vol. 84(4), pp. 573-586, Elsevier, 2011.

[Daigle and Roychoudhury, 2010] M. Daigle and I. Roychoudhury. Qualitative Event-based Diagnosis: Case Study on the Second International Diagnostic Competition. In Proceedings of 21st International Workshop on Principles of Diagnosis, Portland, OR, 2010.

[Daigle et al., 2011] M. Daigle, A. Bregon, and I. Roychoudhury. Qualitative Event-based with Possible Conflicts: Case Study on the Third International Diagnostic Competition. In Proceedings of 22nd International Workshop on Principles of Diagnosis, Munich, Germany, 2011.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. Artificial Intelligence, 32(1):97-130, 1987.

[de Kleer, 2008] J. de Kleer. An Improved Approach for Generating Max-Fault Min-Cardinality Diagnoses. In Proceedings of 19th International Workshop on Principles of Diagnosis, Blue Mountains, Australia, 2008.

[de Kleer, 2011] J. de Kleer. Hitting Set Algorithms for Model-based Diagnosis. In Proceedings of 22nd International Workshop on Principles of Diagnosis, Munich, Germany, 2011.

[Feldman et al., 2008] A. Feldman, G. Provan, A. van Gemund. Computing observation vectors for Max-Fault Min-Cardinality diagnoses. In Proc. AAAI'08, pp. 919–924.

[Feldman et al., 2010] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, A. van Gemund. Empirical Evaluation of Diagnostic Algorithm Performance Using a Generic Framework. In International Journal of Prognostics and Health Management, Vol. 1 (2), 2010.

[Gonzalez-Sanchez et al., 2011] A. Gonzalez-Sanchez, R, Abreu, H.G. Gross, A.J.C. van Gemund. Prioritizing Tests for Fault Localization through Ambiguity Group Reduction. In Proceedings of the 26th International Conference on Automated Software Engineering (ASE'11). Lawrence, KA, November 2011.

[Kurtoglu et al., 2009a] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman. Towards a Framework for Evaluating and Comparing Diagnosis Algorithms. In Proceedings of 20th International Workshop on Principles of Diagnosis, Stockholm, Sweden, 2009.

[Kurtoglu et al., 2009b] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, A. Feldman. First International Diagnosis Competition – DXC'09. In Proceedings of 20th International Workshop on Principles of Diagnosis, Stockholm, Sweden, 2009.

[Mange et al., 2011] J. Mange, D Daniszewski, and A. Dunn. Artificial Immune Systems for Diagnostic

Classification Problems. In Proceedings of 21st International Workshop of Principles of Diagnosis, Munich, Germany, 2011.

[Mosterman and Biswas, 1999] P. J. Mosterman and G. Biswas. Diagnosis of Continuous Valued Systems in Transient Operating Regions. In IEEE Trans. on Systems, Man and Cybernetics, vol. 29, no. 6, pp. 554-565, Nov. 1999.

[Narasimhan and Brownston, 2007] S. Narasimhan and Lee Brownston. HyDE – A General Framework for Stochastic and Hybrid Model-based Diagnosis. In Proceedings of 18th International Workshop on Principles of Diagnosis, Nashville, TN, 2007.

[Poll et al., 2010] S. Poll, J. de Kleer, A. Feldman, D. Garcia, T. Kurtoglu, and S. Narasimhan. Second International Diagnostic Competition – DXC'10. In Proceedings of 21st International Workshop on Principles of Diagnosis, Portland, OR, 2010.

[Poll et al., 2011] S. Poll, J. de Kleer, R. Abreu, M. Daigle, A. Feldman, D. Garcia, A. Gonzalez-Sanchez, T. Kurtoglu, S. Narasimhan, and A. Sweet. Third International Diagnostic Competition – DXC'11. In Proceedings of 22st International Workshop on Principles of Diagnosis, Murnau, Germany, 2011.

[Pulido and Alonso-Gonzalez, 2004] B. Pulido and C. Alonso-Gonzalez. Possible conflicts: a compilation technique for consistency-based diagnosis. IEEE Trans. Syst. Man Cy. B., 34(5):2192–2206, 2004.

[Siddiqi and Huang, 2007]. S. Siddiqi and J. Huang. Hierarchical Diagnosis of Multiple Faults. In Proc. IJCAI'07, pp. 581–586.

Qualitative Event-based Diagnosis with Possible Conflicts: Case Study on the Fourth International Diagnostic Competition*

Matthew Daigle[1] and **Indranil Roychoudhury**[2] and **Anibal Bregon**[3]
[1]NASA Ames Research Center, Moffett Field, CA, 94035, USA
e-mail: matthew.j.daigle@nasa.gov
[2]SGT Inc., NASA Ames Research Center, Moffett Field, CA, 94035, USA
e-mail: indranil.roychoudhury@nasa.gov
[3]Department of Computer Science, University of Valladolid, Valladolid, 47011, Spain
e-mail: anibal@infor.uva.es

Abstract

The objective of the International Diagnostic Competition is to provide a platform for evaluating how different diagnostic algorithms perform and compare to one another when applied to the same problem. This paper describes three model-based diagnosis algorithms entered into the Fourth International Diagnostic Competition. We focus on the first diagnostic problem of the industrial track of the competition in which a diagnosis algorithm must detect, isolate, and identify faults in an electrical power distribution testbed in order to provide abort recommendations when warranted. We present here a general fault isolation framework that encompasses three algorithms, each of which use different residual sets for fault isolation; one based on the global system model, one based on minimal submodels computed using Possible Conflicts, and one based on the combination of the former two residual sets. We describe, compare, and contrast the three algorithms in terms of practical implementation and their diagnosis results.

1 Introduction

In this paper, we present a model-based, qualitative, event-based fault diagnosis scheme entered into the Fourth International Diagnostic Competition (DXC'13). The competition allows for a comparative study of different diagnostic approaches, and includes multiple diagnostic problems. Different diagnostic algorithms applied to the same diagnostic problem are compared to one another and ranked in terms of metrics developed in [Poll *et al.*, 2010; 2011]. In this work, we focus on diagnostic problem I (DPI) of the industrial track of the competition, which consists of fault diagnosis and recovery for a subset of the Advanced Diagnosis and Prognosis Testbed (ADAPT) [Poll *et al.*, 2007], called ADAPT-Lite, which is an electrical power distribution system. Our diagnosis scheme has three instantiations, the first two being QED (Qualitative Event-based Diagnosis) and QED-PC (QED with Possible Conflicts [Pulido and Alonso-González, 2004]), both of which were previously submitted to the Third International Diagnostic Competition (DXC'11) [Daigle *et al.*, 2012], and have been improved and updated for this new edition of the competition. In addition to these two algorithms, we present a new entry in this year's diagnostic competition, QED-PC++, which may be considered as a combination of QED and QED-PC.

QED extends the TRANSCEND qualitative diagnosis scheme described in [Mosterman and Biswas, 1999]. In this scheme, fault isolation is obtained through analysis of the transients produced by faults, manifesting as deviations in observed behavior from predicted nominal behavior. TRANSCEND was extended by including relative residual orderings, which provide a partial ordering of measurement deviations for different faults, leading to an enhanced event-based fault isolation scheme [Daigle *et al.*, 2009; 2007]. TRANSCEND deals only with abrupt faults, so in [Daigle *et al.*, 2012] we incorporated methods for incipient and intermittent fault isolation and identification.

Whereas QED uses a global model of the system for residual generation, the second algorithm, QED-PC, uses the Possible Conflicts (PCs) diagnosis approach (presented in [Pulido and Alonso-González, 2004]), in which residuals are generated from minimal single-output submodels. This approach decomposes the global system model into minimal submodels containing sufficient analytical redundancy to generate fault hypotheses from observed measurement deviations. In this work, we use the qualitative fault isolation framework of QED to perform residual analysis, using the PC-based residuals. For fault identification, we use minimal parameter estimators computed from PCs for each faulty parameter as proposed in [Bregon *et al.*, 2012].

The third algorithm, named QED-PC++, can be seen as the combination of the previous two diagnosis schemes. QED-PC++ uses the residual sets used by QED (computed using the global system model) and QED-PC (computed from the PCs) to form a diagnosis scheme which improves upon QED and QED-PC individually. In the previous competition, we found that QED excelled at isolating nonsensor faults, but needed ad hoc fault isolation rules in order to isolate sensor faults without the aid of fault identification. In contrast, for QED-PC, we found that it excelled at isolating sensor faults, but required ad hoc fault isolation rules to isolate nonsensor faults. By using residuals from both the global model and the PCs, we eliminate the weaknesses of the individual residual sets and obtain improved diagnosability for QED-PC++ over QED and QED-PC individually.

The paper is organized as follows. First, Section 2 overviews the general diagnosis approach. Section 3 dis-

*M. Daigle and I. Roychoudhury's funding for this work was provided by the NASA System-wide Safety and Assurance Technologies (SSAT) Project. A. Bregon's funding for this work was provided by the Spanish MCI TIN2009-11326 grant.

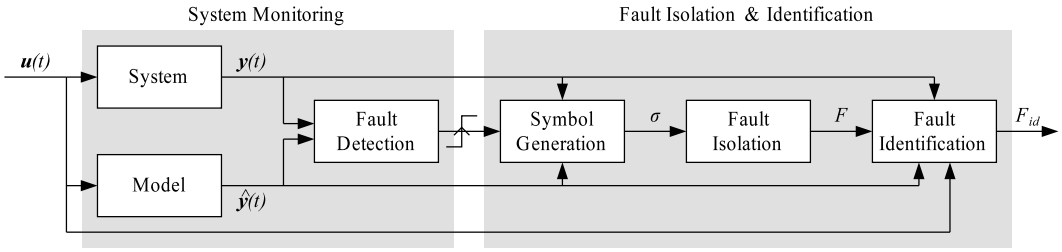


Figure 1: Diagnosis architecture.

cusses diagnosability within our fault isolation framework. Section 4 describes three diagnosers that are instantiated from the general diagnosis approach. Section 5 presents diagnosis results, and Section 6 concludes the paper.

2 Diagnosis Approach

Our diagnosis approach uses the model of the system, and performs the tasks of (*i*) fault detection, (*ii*) fault isolation, and (*iii*) fault identification. The diagnosis architecture is shown in Fig. 1, and reflects the implementation of the three algorithms. Next, we briefly introduce each of the main blocks within our architecture. Additional details on each module may be found in [Daigle *et al.*, 2012].

2.1 System Modeling

ADAPT-Lite consists of relays, circuit breakers, AC and DC resistive loads, a fan, a battery, and a DC to AC inverter, along with 11 sensors measuring current, voltage, relay position, fan speed, and battery temperature (see [Poll *et al.*, 2011]). Our diagnosis approach is model-based, requiring a model of both nominal and faulty behavior for use throughout the diagnosis process. The three algorithms implement the nominal model in a different way. For QED, the nominal model is a global model of the system, $\mathcal{M}$, and its inputs are those of the global system. For QED-PC, the nominal model is composed of a set of 11 minimal submodels computed from PCs, with each submodel $\mathcal{M}_i$ estimating the value of sensor i using a subset of the system measurements as input variables. QED-PC++ uses both the global model and the 11 submodels, thus having 22 total outputs and 22 residuals.

2.2 Fault Detection

The three algorithms use the same approach for fault detection. Each residual is assigned a fault detector. For each output $y(t)$, we define the residual as $r(t) = y(t) - \hat{y}(t)$, where $\hat{y}(t)$ is the model-predicted output signal. As previously described, for QED, $\hat{y}(t)$ is computed using the global model, for QED-PC, it is computed using the corresponding PC, and for QED-PC++, it is computed using each. Fault detection works by applying a Z-test to the residual values. The Z-test detects statistically significant nonzero residual signals, which indicate the presence of a fault.

The real sensor data contains some intermittent spikes which are not to be classified as faults. In order to avoid false alarms and incorrect fault isolation due to data spikes, we implement a median filter, which takes the last three values of a given sensor and reports the median of these values as the current sensor value. As a result, fault detection will be delayed and its sensitivity decreased, however this is preferred over false alarms and misdiagnoses.

2.3 Fault Isolation

In our proposal, we use a qualitative diagnosis methodology that isolates faults based on the transients they cause in system behavior, manifesting as deviations in residual values [Mosterman and Biswas, 1999]. The transients produced by faults are abstracted using qualitative + (increase), − (decrease), and `0` (no change) values and `N` (zero to nonzero), `Z` (nonzero to zero), and `X` (no discrete change) values to form *fault signatures*. Fault signatures represent these measurement deviations from nominal behavior as the immediate (discontinuous) change in magnitude, the first nonzero derivative change, and discrete zero/nonzero value changes in the measurement from the estimate caused by discrete faults. These symbols are computed from the residuals using symbol generation, as described in [Daigle and Roychoudhury, 2010]. In addition we use the relative order of residual deviation, termed relative residual orderings, to isolate faults [Daigle *et al.*, 2007]. Fault isolation fundamentals for the algorithms will be detailed in Section 3.

In addition to qualitative fault isolation, some heuristic fault isolation rules are introduced to improve fault isolation times and overall diagnosability. For example, if a relay fault occurs, we should observe a change in the residual for its corresponding sensor almost immediately. Most rules are of the form where we give an emprical time limit to an expected deviation for a fault, such that if the deviation is not observed, we eliminate that fault from the candidate list. Specific rules are provided in [Daigle *et al.*, 2012]. For QED-PC++, due to its improved diagnosability from the combined residual set, most rules are eliminated. For example, for QED we need a rule to rule out nonsensor faults if after a significant amount of time only one residual has been observed to deviate; QED-PC++ does not require that rule because multiple PC-based residuals will deviate due to a sensor fault.

2.4 Fault Identification

Fault identification takes the set of isolated faults, and computes the fault parameters for each, producing a new fault set F_{id} including this information. Identification is initiated immediately after the initial set of fault candidates is produced after fault detection. An identification routine is run for each fault candidate, which updates at each time step. All of the algorithms use minimal submodels for fault identification, except for sensor faults in which QED and QED-PC++ use the global model.

2.5 Recovery

At the end of the scenario, the decision whether to abort or continue the mission must be made. The fault identification module computes a candidate set F_{id}, with each $f \in F_{id}$

being defined by the component, its fault mode, and the associated fault parameters. The oracle is viewed as a function $O(f)$ which, for a given fault, computes a recommended set of commands C. For DPI, either $C = \{abort\}$ or $C = \varnothing$.

Each command set has an associated cost. The cost is zero when the correct command is chosen. If the algorithm recommends $abort$ when the mission should be continued, the associated cost is that of the mission (25). If the algorithm recommends to continue when it should have been aborted, the associated cost is that of the mission and the vehicle (125). Therefore, we take the conservative approach and recommend $abort$ if $O(f) = \{abort\}$ for at least one $f \in F_{id}$. In the case that a fault was detected but all candidates were eliminated, then one may assume either a false positive, or a true positive with incorrect fault isolation. We assume the latter, and in this case, we again conservatively recommend an abort.

3 Diagnosability

For a given model, through the qualitative fault isolation framework we can generate a set of fault signatures and relative residual orderings, which form, based on a set of residuals, the diagnostic information of the qualitative approach. Using the general model decomposition framework described in [Roychoudhury *et al.*, 2013], we can generate, given a global system model, a number of independent submodels for the purposes of residual generation. A submodel is defined by its subset of the variables and constraints of the global model. For defining residuals, the outputs of the submodels are the important variables. Given a set of (measured) outputs, we can generate a minimal submodel. For a set of m total outputs, we can define m single-output submodels, $\binom{m}{2}$ double-output submodels, and so on, and one submodel with all m outputs (i.e., the global model). For a system with m measurements the number of possible submodels is $2^m - 1$, and the number of unique residuals over all possible submodels is $m \times 2^{m-1}$.

In our qualitative fault isolation framework, deviations in residuals resulting from faults are abstracted into qualitative symbols that can be reasoned over. These symbolic abstractions are termed fault signatures [Mosterman and Biswas, 1999; Daigle *et al.*, 2009]. In order to define diagnosability, we must first formalize the fault isolation framework. The framework was originally presented in [Daigle *et al.*, 2009], and we extend and generalize it here to account for residuals from a set of submodels.

Definition 1 (Fault Signature)**.** A *fault signature* for a fault f and residual r, denoted by $\sigma_{f,r}$, is set of symbols representing potential qualitative changes in r caused by f at the point of the occurrence of f. The set of fault signatures for f and r is denoted as $\Sigma_{f,r}$.

The temporal order of the residual deviations can also be used as discriminatory information. The temporal order of residual deviations for a given model, termed *relative residual orderings*, are based on the intuition that fault effects will manifest in some parts of the system before others, and can be computed based on analysis of the transfer functions from faults to residuals [Daigle *et al.*, 2007].

Definition 2 (Relative Residual Ordering)**.** If fault f always manifests in residual r_i before residual r_j, then we define a *relative measurement ordering* between r_i and r_j for fault f, denoted by $r_i \prec_f r_j$. We denote the set of all residual orderings for f as $\Omega_{f,R}$.

Signatures and orderings can be generated by manual analysis of the system model, by simulation, or automatically from certain types of models, e.g., as presented in [Daigle, 2008]. Together, they establish an event-based form of diagnostic information. For a given fault, the combination of all fault signatures and residual orderings yields all the possible ways a fault can manifest in the residuals. Each of these possibilities is a *fault trace* [Daigle *et al.*, 2009].

Definition 3 (Fault Trace)**.** A *fault trace* for a fault f over residuals R, denoted by $\lambda_{f,R}$, is a sequence of fault signatures, of length $\leq |R|$ that includes, for every $r \in R$ that will deviate due to f, a fault signature $\sigma_{f,r}$, such that the sequence of fault signatures satisfies $\Omega_{f,R}$.

We group the set of all fault traces into a *fault language*:

Definition 4 (Fault Language)**.** The *fault language* of a fault $f \in F$ with residual set R, denoted by $L_{f,R}$, is the set of all fault traces for f over the residuals in R.

In our diagnosis framework, distinguishability between faults is characterized using fault traces and languages.

Definition 5 (Distinguishability)**.** Given a residual set, R, a fault f_i is *distinguishable* from a fault f_j, if there does not exist a pair of fault traces $\lambda_{f_i,R} \in L_{f_i,R}$ and $\lambda_{f_j,R} \in L_{f_j,R}$, such that $\lambda_{f_i} \sqsubseteq \lambda_{f_j}$.

One fault will be distinguishable from another fault if it cannot produce a fault trace that is a prefix[1] (denoted by $\sqsubseteq$) of a trace that can be produced by the other fault. If this is not the case, then when that trace manifests, the first fault cannot be distinguished from the second.

Distinguishability is used to define the diagnosability of a diagnosis model under a given fault isolation framework. A diagnosis model is an abstraction of a system model with only diagnosis-relevant information.

Definition 6 (Diagnosis Model)**.** A *diagnosis model* $\mathcal{S}$ is a tuple $(F, R, L_{F,R})$, where $F = \{f_1, f_2, \ldots, f_n\}$ is a set of faults, R is a set of residuals, and $L_{F,R} = \{L_{f_1,R}, L_{f_2,R}, \ldots, L_{f_n,R}\}$ is the set of fault languages.

If a diagnosis model is diagnosable, then we can make guarantees about the unique isolation of every fault in the diagnosis model.

Definition 7 (Diagnosability)**.** A diagnosis model $\mathcal{S} = (F, R, L_{F,R})$ is *diagnosable* if and only if $(\forall f_i,\ f_j \in F) f_i \neq f_j \implies f_i \not\sim_R f_j$.

If $\mathcal{S}$ is diagnosable, then every pair of faults is distinguishable using the residual set R. Hence, we can uniquely isolate all faults of interest. If $\mathcal{S}$ is not diagnosable, then ambiguities will remain after fault isolation, i.e., after all possible fault effects on the residuals have been observed.

4 Diagnosers

A qualitative fault diagnoser, in our framework, is then defined by the set of faults and a set of residuals. From the large residual space, any subset of residuals can, in theory, be selected. In practice, there is much redundant information over the residuals, and, therefore, only a subset are required to achieve an appropriate level of diagnosability. The three diagnostic algorithms we develop are based on three different qualitative fault diagnosers with different diagnosability properties.

[1] A fault trace λ_i is a prefix of fault trace λ_j if there is some (possibly empty) sequence of events λ_k that can extend λ_i such that $\lambda_i \lambda_k = \lambda_j$.

Table 1: Selected Fault Signatures for the QED algorithm for ADAPT-Lite

Fault	E240	E242	IT281	IT267	ST516
AC483 $\Delta p > 0$	+0X	+0X	+0X	−0X	00X
DC485 $\Delta p > 0$	+0X	+0X	−0X	00X	00X
E240 $\Delta p > 0$	+0X	00X	00X	00X	00X
E240 $m > 0$	0+X	00X	00X	00X	00X
E240 $\mu_{\Delta p} > 0$	+0X	00X	00X	00X	00X
EY244 stuck open	+0X	−0Z	−0Z	−0Z	0−X
FAN416 underspeed	+0X	+0X	00X	−0X	−0X

Table 2: Selected Fault Signatures for the QED-PC algorithm for ADAPT-Lite

Fault	E240	E242	IT281	IT267	ST516
AC483 $\Delta p > 0$	00X	00X	00X	−0X	00X
DC485 $\Delta p > 0$	00X	00X	−0X	00X	00X
E240 $\Delta p > 0$	+0X	−0X	00X	00X	00X
E240 $m > 0$	0+X	0−X	00X	00X	00X
E240 $\mu_{\Delta p} > 0$	+0X	−⋆X	00X	00X	00X
EY244 stuck open	00X	−0Z	00X	00X	00X
FAN416 underspeed	00X	00X	00X	−0X	−0X

4.1 QED

The first algorithm, QED, uses the set of residuals defined from the global model [Daigle and Roychoudhury, 2010], and is based on the extended Transcend approach [Daigle *et al.*, 2009]. Fault signatures for selected faults are shown in Table 1.

A diagnosability analysis reveals several instances where one fault cannot be distinguished from another. The first set of indistinguishable faults is for the four pairs of faults that produce exactly the same quantitative behavior on the given measurements: failures in CB262 and INV2, failures in EY281 and DC485, failures in EY272 and AC483, and failures in EY275 and FAN416. Therefore, based on only qualitative information they cannot be distinguished either. The second set of indistinguishable faults is between offset and intermittent faults, which produce the same initial transients. It is only through fault identification that they can be distinguished. The third set of inidistinguishable faults deals with sensors. For example, consider an offset in E240 (see Table 1). An abrupt increase will be observed in E240, and at this point, an offset in AC83, DC483, EY244 stuck, and FAN416 underspeed are all consistent. We have to wait infinitely long to verify that no other residuals will deviate in order to eliminate these faults.

So, when sensor faults occur, qualitative fault isolation alone cannot uniquely distinguish them, and fault identification is needed to resolve the ambiguities. Here, we take advantage of knowledge that if a nonsensor fault occurs, it should cause deviations in multiple residuals within finite time. Therefore, we create a heuristic fault isolation rule that implements this idea, e.g., if after 60 seconds only one residual has deviated, we eliminate all nonsensor faults. In this way, we can uniquely isolate sensor faults without fault identification, and improve fault isolation time.

4.2 QED-PC

The second algorithm, QED-PC, uses the set of residuals defined from the minimal single-output submodels [Daigle *et al.*, 2012], and is based on the Possible Conflicts approach [Pulido and Alonso-González, 2004], augmented with the qualitative fault isolation framework [Bregon, 2010]. Signatures for selected faults are shown in Table 2.

The main advantage of the PC-based residuals is that, like ARRs and MSOs, they decouple faults from residuals, so faults affect only a subset of the residuals [Armengol *et al.*, 2009]. In most cases, the decoupling increases diagnosability. For example, if each fault affects only a single unique residual, the system is diagnosable even in the multiple fault case. For QED-PC, sensor faults can now be easily distinguished, because they affect mutliple residuals. The reason is that sensed values are used as inputs, so a sensor fault will cause a deviation in the residual for the PC that computes the output as well as any PCs that use the sensor's values as an input. However, now there is a diagnosability problem with nonsensor faults. For example, a fault in AC483 affects only a single residual. So, like sensor faults with QED, we have to wait infinitely long to ensure it is not some other fault that produces a deviation in that residual.

Similarly to QED, we can improve diagnosability by introducing some heuristic fault isolation rules for these faults. In this case, we can assume that sensor faults would produce deviations on more than one residual in finite time. This will allow faults like those in AC483 and DC485 to be distinguished without resorting to fault identification, and improve fault isolation times.

4.3 QED-PC++

The diagnosability analyses of QED and QED-PC reveal the weaknesses of the individual residual sets. When sensor faults occur, QED cannot distinguish them from nonsensor faults. When some nonsensor faults occur, on the other hand, QED-PC cannot distinguish them from sensor faults. This suggests that an algorithm using the combined residual sets of QED and QED-PC can resolve these ambiguities and lead to improved fault isolation. This is the residual set used by the third algorithm, QED-PC++.

We can see now that the diagnosability weaknesses of QED and QED-PC are resolved by a combined residual set. When a sensor fault occurs, only one global model residual will deviate, but multiple PC-based residuals will deviate. For the indistinguishable nonsensor faults in QED-PC, only one PC-based residual will deviate but multiple global model residuals will deviate. This eliminates the need for most of the heuristic fault isolation rules and improves fault isolation times.

5 Results

In order to demonstrate the differences between the diagnosers, we describe a demonstration scenario in which a drift fault in E242 is injected at 175 s with a slope of 0.075. Measured and predicted values for relevant outputs are shown in Figs. 2–5. QED detects the fault at 176.7 s, with an increase in the E242 residual. The possible faults are in AC483, CB262, DC485, E242, EY260, EY272, EY275, EY284, FAN416 (failed off or underspeed), and INV2. At 176.9 s the stuck mode of E242 is eliminated since consecutive measurements were of different values. At 177.1 s the discrete change symbol is computed as X, which does not change the candidate list. At 179.2 s, the slope of E242

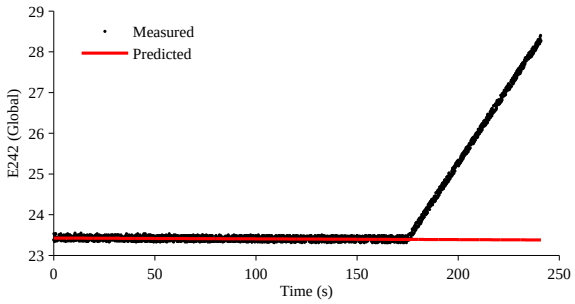


Figure 2: Measured and predicted values of E242 (global model) for E242 drift fault.

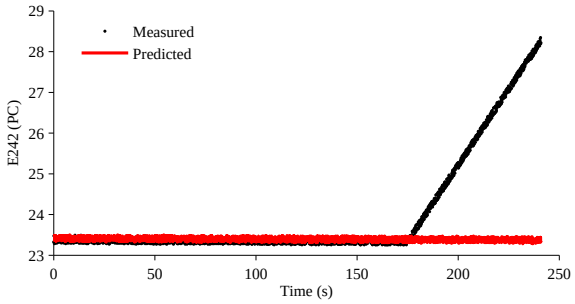


Figure 3: Measured and predicted values of E242 (PC) for E242 drift fault.

is computed as +, eliminating all candiates but AC483 resistance drift, DC485 resistance drift, and E242 drift. At 220 s, since only one residual had deviated, it is concluded to be a sensor fault and E242 drift is isolated. The injection time is computed as 169.8 s and the magnitude as 0.069. The recommended action is ∅, which is correct.

QED-PC detects the fault at 178.1 on the PC for E242. The thresholds for the PCs are larger, so fault detection is slower than with QED. The initial candidate list consists only of faults in E240 and E242, since the remaining faults are decoupled from the E242 residual by the PC design. This is in constrast to QED, where most components except sensors were implicated with the first residual deviation. At 178.3, it is determined that neither E240 or E242 are stuck. At 182.1 the discrete change symbol for E242 is computed as X, which does not change the candidate list. At 182.1, a decrease in the E281 PC residual and increase in the IT240 PC residual are detected, isolating the fault to

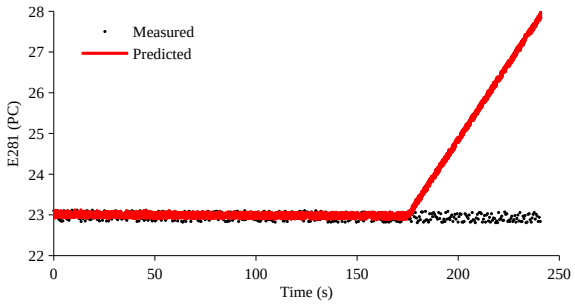


Figure 4: Measured and predicted values of E281 (PC) for E242 drift fault.

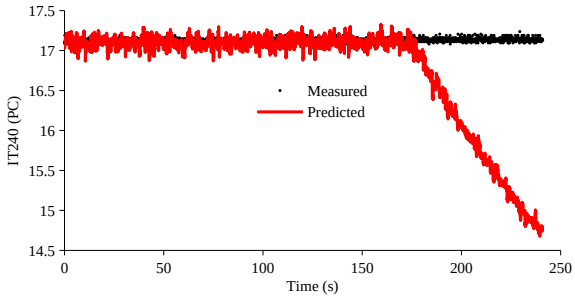


Figure 5: Measured and predicted values of IT240 (PC) for E242 drift fault.

E242 (drift or offset). At 183.1, the slope on E242 is computed as +, therefore isolating E242 drift as the fault. The injection time is computed as 169.7, and the slope as 0.069. The recommended action is ∅.

QED-PC++ detects the fault at 176.7 with the E242 global model residual. The initial candidate list is the same as with QED - faults are in AC483, CB262, DC485, E242, EY260, EY272, EY275, EY284, FAN416 (failed off or underspeed), and INV2. At 176.9 it is determined that E242 is not stuck. At 177.1 the discrete change symbol for E242 is computed as X, and the candidate list remains unchanged. At 178.1, an increase in the residual for the E242 PC is detected, thus eliminating all candidates except for faults in E242. Thus, fault isolation is completed far earlier than QED (about 30 s), and also earlier than QED-PC (about 5 s). The fault is computed as a drift with an injection time of 169.8 and a slope of 0.069. The recommended action is ∅.

Overall results are shown in Table 3. The metrics consist of the mean time to detect faults M_{fd}, the mean false negative rate M_{fn}, the mean false positive rate M_{fp}, the detection accuracy M_{da}, the mean time to isolate faults M_{fi}, the number of classification errors M_{err}, the mean CPU time M_{cpu}, the mean peak memory usage M_{mem}[2], and the overall recovery cost M_{rc}. Both QED and QED-PC are improved from their performance in DXC'11 (see [Poll *et al.*, 2011]). One source of previous errors was data spikes, which caused false alarms when occurring before the fault and incorrect symbol generation when occurring after the fault. The solution was to use a median filter over 3 samples. As a result, residual thresholds had to be increased slightly for some sensors. Other changes in included minor fixes to the fault signatures, and further tuning to residual thresholds to decrease sensitivity and avoid false alarms and incorrect symbol generation.

Overall, all algorithms do very well. QED and QED-PC++ have errors on the same scenarios. There are 12 errors due to the cases for the nondistinguishable faults (e.g., DC485 failing vs its relay failing). The remaining errors are due to incorrect fault mode identification (e.g., identifying as intermittent instead of drift) and missed detections. The missed detections are acceptable, because the faults are small enough that the correct action is ∅. The incorrect mode identification scenarios also did not result in a bad recommendation. The one scenario that did was one where the identified fault parameters were off just enough so that

[2]Over multiple runs, CPU time and memory usage will vary and within the statistical deviation, M_{cpu} and M_{mem} can be considered equivalent for all three algorithms.

Table 3: QED Diagnosis Results

DA	M_{fd} (s)	M_{fn}	M_{fp}	M_{da}	M_{fi} (s)	M_{err}	M_{cpu} (ms)	M_{mem} (kb)	M_{rc}
QED	9.38	0.0152	0.0	0.987	125.61	19	22.72	7675	25
QED-PC	14.66	0.025	0.0	0.978	127.70	37	23.70	7743	275
QED-PC++	9.32	0.0152	0.0	0.987	124.94	19	24.49	7835	25

an abort was recommended when the correct action was $\varnothing$.

QED-PC had similar errors, and also some additional ones in which the DC485 and IT281 faults were confused. This is a difficult situation to correct, because the thresholds were difficult to tune. DC485 can only be uniquely isolated if only IT281 deviates, so if we see additional deviations we conclude it is an IT281 fault. However, sometimes IT281 did not cause large enough deviations in other residuals so it was misdiagnosed as DC485. These scenarios did not result in a bad recommendation, though. In two cases, QED-PC recommended $\varnothing$ when the correct action was to abort. In one case, an ST516 fault was misdiagnosed as an E265 fault. This error is due to threshold sensitivities (for more discussion, see [Daigle *et al.*, 2012]). In the other case, an ST516 intermittent offset fault was misdiagnosed as an offset fault, for which the identified offset was not enough to trigger an abort recommendation.

6 Conclusions

In this paper, we presented three diagnostic algorithms based on a common qualitative fault isolation framework. The algorithms differ in which residual generators are used for fault detection and isolation. We showed that a combination of global model and PC-based residuals offers the best diagnosability and fault isolation performance, as confirmed with experimental results.

Although the diagnostic algorithms do fairly well, there are several areas that can be improved. For one, it was found that tuning thresholds for slope generation was very difficult. It would most likely be easier to avoid using that symbol, and rely only on fault identification to sort through the possible component modes. That would make the algorithms easier to use up front and eliminate the chances of incorrect symbol generation. Mode identification was also an issue, and it was difficult to tune the discrete-valued tests that determined which fault mode was present. A more general, probability-based approach would be more favorable, that, for instance, ranks the possible modes according to how best they fit the data.

References

[Armengol *et al.*, 2009] J. Armengol, A. Bregon, T. Escobet, E. Gelso, M. Krysander, M. Nyberg, X. Olive, B. Pulido, and L. Travé-Massuyès. Minimal structurally overdetermined sets for residual generation: A comparison of alternative approaches. In *Proc. of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, Barcelona, Spain, 2009.

[Bregon *et al.*, 2012] A. Bregon, G. Biswas, and B. Pulido. A decomposition method for nonlinear parameter estimation in TRANSCEND. *IEEE Trans. Syst. Man. Cy. Part A*, 42(3):751–763, 2012.

[Bregon, 2010] A. Bregon. *Integration of FDI and DX Techniques within Consistency-based Diagnosis with Possible Conflicts*. PhD thesis, University of Valladolid, 2010.

[Daigle and Roychoudhury, 2010] M. Daigle and I. Roychoudhury. Qualitative event-based diagnosis: Case study on the second international diagnostic competition. In *Proceedings of the 21st International Workshop on Principles of Diagnosis*, pages 371–378, October 2010.

[Daigle *et al.*, 2007] M. J. Daigle, X. D. Koutsoukos, and G. Biswas. Distributed diagnosis in formations of mobile robots. *IEEE Trans. on Robotics*, 23(2):353–369, April 2007.

[Daigle *et al.*, 2009] M. J. Daigle, X. Koutsoukos, and G. Biswas. A qualitative event-based approach to continuous systems diagnosis. *IEEE Trans. on Control Systems Technology*, 17(4):780–793, July 2009.

[Daigle *et al.*, 2012] M. Daigle, A. Bregon, and I. Roychoudhury. Qualitative event-based diagnosis with possible conflicts applied to spacecraft power distribution systems. In *Proceedings of the 8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 265–270, August 2012.

[Daigle, 2008] M. Daigle. *A Qualitative Event-based Approach to Fault Diagnosis of Hybrid Systems*. PhD thesis, Vanderbilt University, 2008.

[Mosterman and Biswas, 1999] P. J. Mosterman and G. Biswas. Diagnosis of continuous valued systems in transient operating regions. *IEEE Trans. on Systems, Man and Cybernetics, Part A*, 29(6):554–565, 1999.

[Poll *et al.*, 2010] S. Poll, J. de Kleer, A. Feldman, D. Garcia, T. Kurtoglu, and S. Narasimhan. Second international diagnostics competition – DXC'10. In *Proceedings of the 21st International Workshop on Principles of Diagnosis*, October 2010.

[Poll *et al.*, 2011] S. Poll, J. de Kleer, R. Abreau, M. Daigle, A. Feldman, D. Garcia, A. Gonzalez-Sanchez, T. Kurtoglu, S. Narasimhan, and A. Sweet. Third international diagnostics competition – DXC'11. In *Proc. of the 22nd International Workshop on Principles of Diagnosis*, pages 267–278, October 2011.

[Poll *et al.*, 2007] S. Poll *et al.* Evaluation, selection, and application of model-based diagnosis tools and approaches. In *AIAA Infotech@Aerospace 2007 Conference and Exhibit*, May 2007.

[Pulido and Alonso-González, 2004] B. Pulido and C. Alonso-González. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 34(5):2192–2206, 2004.

[Roychoudhury *et al.*, 2013] I. Roychoudhury, M. Daigle, A. Bregon, and B. Pulido. A structural model decomposition framework for systems health management. In *Proc. of the 2013 IEEE Aerospace Conf.*, March 2013.

Author Index

Proceedings of
The 24th International Workshop on Principles of Diagnosis

Edited by Alexander Feldman, Meir Kalech, and Gregory Provan

The International Workshop on Principles of Diagnosis (DX) is an annual event that started in 1989 and is rooted in the Artificial Intelligence (AI) community. Papers presented at the workshop cover a variety of theories, principles, and computational techniques for diagnosis, monitoring, testing, reconfiguration, fault- adaptive control, and repair of complex systems. Applications of these theories, principles, and techniques to industry-related disciplines and other real-world problems are also important topics of the workshop.

Like the previous workshops in this series, DX-2013 encourages the interactions and the exchange of theories, techniques, applications, and experiences amongst researchers and practitioners from diverse backgrounds – artificial intelligence, control theory, verification, software and systems engineering, and other related areas – who share an interest in different aspects of diagnosis, and the related fields of testing, reconfiguration, maintenance, prognosis, and fault-adaptive control.

DX is a lively forum that has traditionally adopted a single track program with a limited number of participants in order to promote detailed technical exchange and debate while at the same time making efforts to develop synergistic approaches to solving real-world problems.